

Forensics Enabled Secure Mobile Computing System For Enterprises

By
Jayaprakash Govindaraj

Under the Supervision of

Dr. Gaurav Gupta, and
Dr. Donghoon Chang



INDRAPRASTHA INSTITUTE *of*
INFORMATION TECHNOLOGY **DELHI**

May 2018

Forensics Enabled Secure Mobile Computing System For Enterprises

By
Jayaprakash Govindaraj

Submitted in partial fulfillment of the requirements for the degree of
Doctor of Philosophy

to



INDRAPRASTHA INSTITUTE *of*
INFORMATION TECHNOLOGY **DELHI**

May 2018

Certificate

This is to certify that the thesis titled “**Forensics Enabled Secure Mobile Computing System For Enterprises**” being submitted by **Jayaprakash Govindaraj** to the Indraprastha Institute of Information Technology Delhi, for the award of the degree of Doctor of Philosophy, is an original research work carried out by him under my supervision. In my opinion, the thesis has reached the standards fulfilling the requirements of the regulations relating to the degree.

The results contained in this thesis have not been submitted in part or full to any other university or institute for the award of any degree/diploma.

Dr. Gaurav Gupta

May, 2018
Department of Computer Science
IIIT-Delhi
New Delhi - 110020

Dr. Donghoon Chang

May, 2018
Department of Computer Science
IIIT-Delhi
New Delhi - 110020

Keywords : Digital Forensic Readiness, Mobile Computing Systems, Mobile Security, Machine learning, Digital Forensic Investigation, Mobile Ads, User behavior, Jailbreak, Rooting, Modified Access Change/ Creation Date and Time Stamps (MAC DTS), APK analysis, IPA analysis, Mobile Application security analysis, Automation, Anti forensics, Loadable Kernel Module (LKM), Bring Your Own Device (BYOD), Mobile Application Security, Automation, Mobile applications, Cyber-attacks

Dedication

To my
mother, father & brother

*Their unconditional love and complete support have kept me positive-minded and
focused towards my goals for all these years.*

Acknowledgments

My Advisors and Mentors

First and foremost, I would like to express my sincere gratitude to my advisor Dr. Gaurav Gupta, for his continuous support, for his patience, motivation, emotional support, and invaluable guidance. His enthusiasm for research and optimistic attitude have always inspired and motivated me. His extensive knowledge and experience have been the most significant help for my research. I am incredibly fortunate and equally thankful for all he has done for me.

I also express my sincere gratitude to my co-advisor, Dr. Donghoon Chang. It is an honor for me to have him as my co-advisor. His professionalism, commitment towards research, humility and positive attitude have been a source of inspiration for me during my time at IIIT-Delhi.

My Infosys Mentors

I am grateful to Dr. Srinivas Padmanabhuni for his advice, guidance and encouraging me to pursue Ph.D. at IIIT Delhi. I would like to express my sincere gratitude to Sachidananda Javli, without his support and guidance, my Ph.D. will not have been possible.

Infosys Ltd.

I am grateful to the Infosys Limited, who generously supported me through the Ph.D. Industry sponsorship programme.

My Friends and Colleagues

I would also like to thank all my beloved friends from IIIT-Delhi; especially, Robin Verma, Sumesh Manjunath R, Pandarasamy Arjunan, Monalisa Jena, Sweta Mishra, Anush Sankaran, Monika Gupta, Paridhi Jain and Madhur Hasija for their valuable friendship and absolute support. All of them have a significant contribution in making my Ph.D. life at IIIT-Delhi pleasant as well as memorable.

I am grateful to all of my Infosys colleagues; especially Rashmi, Swapniel, Sakina, Amitesh and Tej for their contributions to my research work.

I also express my regards to all of those who supported me in any respect during the completion of my research.

Examiners

Dr. Kam Pui Chow

Associate Professor

Department of Computer Science

The University of Hong Kong

Prof. Yong Guan

Department of Electrical and Computer Engineering

IOWA State University, Ames, IA, USA

Dr. Emmanuel Shubhakar Pilli

Associate Professor

Department of Computer Science & Engineering

MNIT, Jaipur, India

Abstract

The enterprises are facing constant security threats with the emergence of new mobile computing devices like smartphones, smartwatches, and wearables. The existing digital forensic enabled security solutions are not able to match the pace at which these technological advances are evolving. In case of a security incident, the enterprises fail to perform a subsequent digital forensic investigation since their security systems are not designed with required Digital Forensic Readiness capability. Currently, there are no well defined Digital Forensic Readiness frameworks for mobile computing devices. There are some existing frameworks that provide partial support. However, they do not have a provision to learn from the past security violation occurrences. There is a need for an automated forensically ready and secure solution, which could improve efficiency and productivity, while continuously learning and adapting to new and unforeseen challenges. The current thesis is devoted to the design of forensics enabled secure mobile computing systems for the enterprises. The author has focused on developing a ‘digital forensic readiness and secure’ system, which targets smartphones, smartwatches, and wearables operating in an enterprise environment; while incorporating machine learning capabilities to make it a learning system. The digital forensic readiness solutions include ‘Precognition’, which performs forensic analysis of suspected mobile applications. Precognition also uses machine learning techniques that utilize feature sets which are extracted from decompiled mobile applications, to identify potential security threats. The author has analyzed over 14151 mobile applications and classified vulnerabilities with an accuracy of 94.2%. The second solution, which concentrates on digital forensic readiness at the operating system level, securely preserves date and time stamps of targeted events running in smartphones, smartwatches, and wearables. These timestamps can be used to validate the digital evidence during a subsequent digital forensic investigation for any of the devices mentioned above. As a third contribution towards promoting digital forensic readiness in the mobile computing devices, the author has presented a novel form of forensic analysis technique, which analyzes over 5498 mobile ads to build the user profile which can be applied to identify the suspect who uses a particular device. The security contribution of the thesis includes an automated security analysis solution for identifying potential mobile security threats. Further, a solution ‘SecureRing’ has been proposed for securing the mobile applications to provide an additional layer of protection against attacks. The author has also designed ‘MobiSecureWrap’, which is an automated solution for wrapping mobile application binaries with additional security layer to protect them against potential threats. MobiSecureWrap

recommends secure solutions based on detected security threats to protect the application binaries. The author evaluated over 5121 mobile applications to achieve a solution recommendation accuracy of 95.3%.

Contents

1	INTRODUCTION	1
1.1	Digital Forensic Readiness	1
1.2	Mobile computing systems: security landscape	1
1.3	Motivation	2
1.4	Goals	3
1.5	Contributions	4
1.6	Thesis organization	6
2	BACKGROUND	8
2.1	A brief history of Forensic Readiness enabled systems	8
2.1.1	Forensic readiness for mobile computing systems	11
2.2	Mobile computing systems: security landscape	12
2.2.1	Mobile application security threats	12
2.2.2	Mobile application security analysis: tools and techniques	14
2.2.3	Handling security incidents	16
2.2.4	Security and privacy threats through mobile ads	16
2.3	Summary	17
3	AUTOMATED DIGITAL FORENSIC READINESS SYSTEM FOR MOBILE COMPUTING DEVICES IN ENTERPRISES	19
3.1	Introduction	19
3.2	Precognition solution architecture	21
3.2.1	Precognition work-flow	23
3.3	Precognition prototype implementation	24
3.3.1	Precognition client app	25

3.3.2	Precognition server component	27
3.3.3	Server component: offline and on-device/emulator security analysis	27
3.3.4	Machine learning implementation	31
3.4	Results	36
3.5	Summary	38
4	FORENSIC READINESS FOR IOS OPERATING SYSTEM AND SECURE SOLUTION FOR IOS APPS	41
4.1	Introduction	41
4.2	Proposed solution	43
4.3	Implementation methodology of our solution	43
4.3.1	Securing the apps (using static library)	43
4.3.2	Preserving the date and time stamps (using dynamic library)	44
4.3.3	Static library	44
4.3.4	Dynamic library	44
4.4	Preventing attacks and / or anti-forensics	48
4.4.1	Using static library	48
4.4.2	Using dynamic library	49
4.5	Experiments and results	49
4.6	Case study	52
4.7	Summary	52
5	FORENSIC READINESS FOR ANDROID OPERATING SYSTEM BY PRE-SERVING DATE AND TIMESTAMPS	54
5.1	Introduction	55
5.2	Android v/s iOS preserving timestamps comparison	58
5.3	Tampering of MAC DTS on android phone	58
5.3.1	Attack methodology and/ or anti-forensics	59
5.4	Implementation methodology	61
5.4.1	Algorithm for the LKM module	62
5.4.2	Procedure for compiling the LKM	62
5.4.3	Loading the LKM on phone	63
5.4.4	Denying deletion of log file	63

5.4.5	Uploading the log file	64
5.4.6	Incident handling	65
5.4.7	Anti forensics analysis	65
5.5	Performance implications	66
5.6	Summary	67
6	FORENSIC ANALYSIS OF MOBILE DEVICE ADS TO IDENTIFY USERS	68
6.1	Introduction	69
6.2	Background	71
6.2.1	What information does ads reveal	72
6.2.2	Mobile advertising architecture	72
6.3	Methodology	74
6.4	Ads on Mobile devices	74
6.4.1	iOS ads architecture	74
6.4.2	On Android	75
6.5	Results and inferences	81
6.5.1	Ads category	81
6.5.2	Ads interests	81
6.5.3	User information	81
6.5.4	Inferences	82
6.6	Summary	84
7	SYSTEM AND METHOD TO PROTECT SMART DEVICES APPLICATION BINARIES AGAINST CYBER ATTACKS	86
7.1	Introduction	87
7.2	Proposed solution	89
7.2.1	Trusted dynamic wrapper creation system	89
7.2.2	Auto wrapper module	90
7.2.3	Machine learning module	91
7.2.4	Blockchain for Trust module	92
7.3	Implementation details	92
7.3.1	Dynamic Wrapper creation and Autowrapping	92
7.3.2	Machine learning implementation	96

7.4	Prototype Tool	100
7.5	Results	102
7.5.1	Test results	102
7.5.2	Performance test results	102
7.5.3	Machine learning results	103
7.6	Summary	105
8	Conclusion	106
8.1	Summary	106
8.2	Future work	108
A	Algorithm for logging MAC DTS using LKM	109

List of Figures

1.1	Forensics enabled secure mobile computing system	4
2.1	Forensic readiness time-line till 2014	8
2.2	Forensic readiness time-line 2015-2018	9
3.1	Precognition system solution architecture.	22
3.2	Server component	28
3.3	Lack of binary protection.	29
3.4	Insecure data storage	31
3.5	iOS insecure data storage	31
3.6	Machine learning flowchart	33
3.7	Vulnerabilities distribution by categories	38
3.8	Domains versus types of vulnerabilities	39
3.9	Vulnerabilities distribution	39
3.10	Vulnerabilities distribution by domains	40
4.1	Solution architecture preserving date and timestamps.	46
4.2	Simulating attacks on devices.	50
4.3	Performance benchmark results	50
4.4	MAC DTS logs for an image file	51
4.5	Preventing debug mode attack	51
5.1	Tampering of date and timestamps.	59
5.2	Solution architecture –preserving date and timestamps.	62
5.3	Storing <i>record.log</i> file in internal memory.	64
5.4	Snapshot of the <i>record.log</i> file for camera image ‘IMAG0699.jpg’.	64
5.5	Performance benchmark results on HTC Wildfire	67

6.1	Mobile ads architecture	73
6.2	iOS ads architecture	75
6.3	Cache-ads information	76
6.4	Cookies-ads information	76
6.5	History db ads information	77
6.6	iOS-flowchart	78
6.7	Android-flowchart	78
6.8	Android history db	78
6.9	Android logcat	79
6.10	Android ads architecture	79
6.11	Android ads extraction experimental setup	80
6.12	Ads category distribution	81
6.13	Ads interests distribution	82
6.14	User information based on ads clicked more than 100 times	82
6.15	User information based on ads clicked less than 100 times	83
6.16	User preferences generated through Google	83
7.1	MobiSecureWrap proposed architecture	90
7.2	MobiSecureWrap sequence flow diagram	91
7.3	iOS - Before injecting <i>dylib</i>	93
7.4	iOS - After injecting <i>secureRingFramework</i>	93
7.5	Android - Original Smali files	95
7.6	Android - New Smali file added	95
7.7	Android - Modified manifest file	96
7.8	Machine learning flowchart	97
7.9	Uploading the app and certificate	100
7.10	Library loaded into app binary	102
7.11	NBC with MobiSecureWrap	103
7.12	Performance test results	104

List of Tables

2.1	OWASP Mobile Top 10 Threats	13
3.1	Forensic and security analysis of apps	24
3.2	List of devices utilized	24
3.3	User data backup	26
3.4	Training data set	34
3.5	OWASP top 10 mobile vulnerabilities	35
3.6	Test data set	35
3.7	Accuracy of vulnerabilities predicted	37
4.1	Static library APIs.	45
4.2	Attacks and results	50
5.1	Preserving MACDTS iOS versus Android Operating systems	58
5.2	List of smartphones used in our experiment.	63
5.3	Performance benchmark parameters	66
6.1	Types of mobile ads	71
6.2	Types of mobile ad targeting	71
6.3	List of Geny-motion emulators.	81
7.1	Vulnerabilities list	98
7.2	Training dataset	99
7.3	Test dataset	100
7.4	Solutions list	101
7.5	Accuracy of solutions predicted	104

Chapter 1

INTRODUCTION

In this chapter, the author presents an introduction to Digital Forensic Readiness (DFR) and its objectives. Discuss the mobile computing system security landscape and the attack surface. Highlight the need and motivation for the DFR framework for mobile computing systems (MCS). Discuss the goals for DFR system for MCS and present authors contribution in this thesis.

1.1 Digital Forensic Readiness

Digital Forensic Readiness (DFR) is characterized by the ability of an enterprise to maximize the use of digital evidence while minimizing costs during the digital investigation. There are multiple circumstances which require enterprises to collect and preserve digital evidence before an incident occurs [134]. As per ISO/IEC 27043 [1] readiness is defined as the process of being prepared before the incident occurs for a digital forensic investigation. The Digital forensics readiness is a very active research field that attracted researchers. DFR combines expertise in forensics, hardware and software engineering. The first idea of DFR was proposed by Tan, J. (2001) [134]. In his work, the author defines the objectives and measures of DFR when incorporated would increase the forensic readiness. Following are the DFR objectives[95]:

- *Reduce Time and Cost* required for conducting the digital forensic investigation
- *Reduce Disruption* while collecting and performing digital forensic investigation without impacting the changing environment with emergency technologies

1.2 Mobile computing systems: security landscape

As per Global System for Mobile Communications (GSMA) report 2017 [69], there are over 5 billion unique mobile users in the world and still growing every day. With ever so competitive mobile market, mobile device manufacturers, vendors, mobile carriers, IT service providers are facing fierce competition and constrained to release their devices, platforms, applications,

services at a rapid pace. Mobile devices are progressively being used for performing business proceedings, banking transactions, email communication, used as mobile wallets, storing personal and confidential information. Today whatever we accomplish online on desktops are getting implemented on to the mobile devices.

In the mobile application landscape, security is one of the major concerns. Security incident handling is one of the primary tasks. As per checkpoint mobile security survey report 2014 [31], 64% of the users view Android platform to be riskiest and vulnerable to security threats, followed by Apple/iOS (16%), Windows phone about 16% and Blackberry about 4%. According to the report, the security risk on Android platform was at a peak in 2014 as compared to 49% in 2013 and around 30% in 2012. Insecure web browsing accounted for 61% of the total factors impacting the safety of mobile data. Currently, the mobile OS layer is not designed to be forensic ready.

Mobile attack surface: Mobile attack surface is exposed at four different levels namely, the application level, operating system level, device level, and infrastructure level [16]. At the application layer, the threats are due to vulnerabilities in the native mobile app, mobile web apps, and hybrid mobile apps. At the operating system level, the threats are due to vulnerabilities in the OS like an unpatched OS, insufficient key management, and more. At the device level, the risks are due to the stolen device, denial-of-service (DOS) attacks, depleting mobile device resources like battery power, computing power, and storage. At the infrastructure level, the attacks are due to client misconfiguration, misconfigured access point, Honeypot Access Point, Denial of Service, Jamming signal attack, and more. Traditionally the proposed solutions involve [73] secure application design, security assessments, and security awareness.

1.3 Motivation

Digital forensic Readiness framework for Mobile computing systems is not well defined. Most of the current solutions address post incident scenarios. Following are some of the key challenges and drawbacks with current systems:

- No well-defined enterprise ready practical DFR system for smart devices (smartphones, smartwatches, wearables), current systems have not leveraged machine learning techniques.
- No mobile computing systems with DFR utilizing ISO/IEC 27043: 2015 readiness processes.
- Lack ability to monitor a particular application/folder on the devices.
- DFR data collection is still vast and not optimized. What if the collected evidence has tampered.

- Multiple mobile application architectures and many different application types (native mobile apps, mobile web apps, Hybrid mobile apps), make creating an application-independent forensic readiness solution a challenging task.
- Multiple mobile platforms, different operating systems, and the virtualization possibilities make it challenging to design an OS-independent forensic readiness solution.
- Most of the current literature is focused on Android and not much work on automating iOS application security analysis.
- Current systems have not applied machine learning techniques to create a learning model for identifying vulnerabilities and recommending solutions.
- Lack of protection for applications and data in case of a compromised device.
- Users across the globe are continually resorting to jailbreaking/rooting their devices. Jailbreaking or rooting further aggravates the problem of application security. Once the device is jailbroken/rooted, malicious users and hackers can obtain application's code and data stored on the device, compromising the security.
- Enterprises are allowing their employees to carry their own computing devices to workplaces, called Bring your own device (BYOD) [93]. The fast acceptance and adaptation of BYOD across enterprises at such a rapid pace are further adding to the existing security challenges.
- Often in case of an attack, security and forensics are considered as separate tracks [6]. There is a need for considering forensic readiness and security together to enhance the overall security of the mobile computing systems.

There is the need for a secure and forensically ready system that does not compromise the usability and functionality of the system. The system that can work in an unsecured environment, and still have structure and methods in place to handle any security incidents. In this thesis, authors have addressed some of the challenges and drawbacks and also proposed some of the novel techniques for forensic analysis. Section 1.5 list out authors contributions in this thesis and section 1.6 provides the thesis organisation.

1.4 Goals

The goals of the DFR Framework for Mobile computing systems:

- Maximize the potential use of digital evidence
- Decrease the time and costs of investigations incurred either directly onto the Mobile computing system, or related to Mobile computing system's services

- Reduce interference with and prevent interruption of Mobile computing systems' business processes
- Improve the current level of information systems security of Mobile computing systems
- Creating a model and procedures for realizing these goals

1.5 Contributions

Refer to figure 1.1 for Forensics Enabled Secure Mobile Computing System components. The authors have implemented forensic readiness and security at the application layer, forensic readiness and security at operating system layer, proposed novel forensic analysis techniques to identify the user profile and automated application security threats mitigation, further authors have leveraged machine learning capabilities in all the layers of the system. Implemented DFR and security together to enhance the overall security of the mobile computing systems. The system would help in enhancing the forensic capability of the enterprises and ensure they are well prepared to handle any cyber attack. The main contributions of the present thesis are as follows.

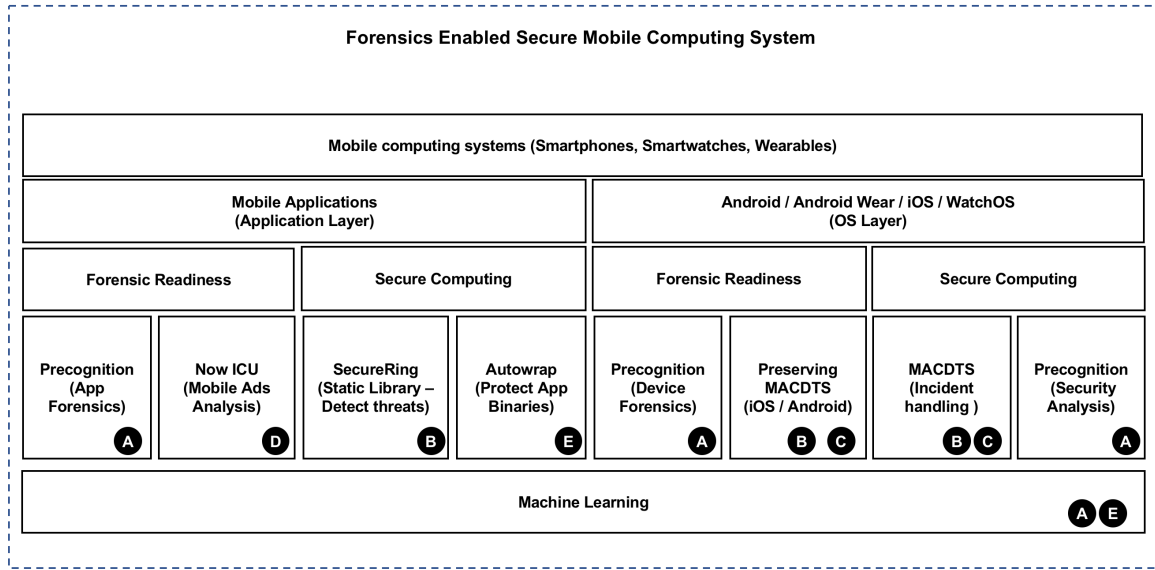


Figure 1.1: Forensics enabled secure mobile computing system

A. The thesis proves the need for automated digital forensics readiness system for mobile computing devices targeting Enterprise environment.

- System to monitor installed apps on a device or an equivalent virtual instance as per the enterprise defined security policies.
- Automated application analysis on devices running Android/AndroidWear/iOS/WatchOS to detect potential security threats.

- Developed machine learning model for predicting such threats to increase the efficiency and accuracy of the system.
- Digital forensic ready system collects data required for digital investigations. In case of breach, forensic investigators would have potential pieces of evidence.
- System was evaluated in a real-world enterprise. Analyzed over 14151 mobile apps, gathered security insights based on industry verticals, app categories, and vulnerabilities distribution. Insights could be beneficial for the enterprises and application owners.

B. Designed and implemented a solution to secure iOS apps. Forensic readiness solution for the iOS operating system by preserving date and time stamps.

- Proposed an approach for any new or existing apps to be secured and made forensics ready, even if the device is jailbroken.
- Static library that could detect security vulnerabilities and take appropriate actions.
- To detect malicious tampering of data by storing an authentic copy of the Modified Access Change/ Creation Date and Time Stamps (MAC DTS) values of the events on a secure location outside the iOS devices.
- Authentic copy of the MAC DTS values can help in offline digital forensics investigations in case of a security incident.

C. Designed and implemented forensic readiness solution for the Android operating system by preserving date and time stamps.

- Demonstrated that the tampering of MAC DTS on Android smartphones is possible (Anti-forensic approach).
- Implemented Loadable Kernel Module (LKM) for the Android operating system to capture the system generated MAC DTS values.
- Designed and implemented a mechanism for preserving date and time stamps in a secure location.
- Proposed a proactive approach to utilize authentic MAC DTS values when a security incident has to be investigated.

D. Proposing a novel approach for identification of user profile by forensic analysis of mobile ads

- Created a system which can track clicked ads live, extract these ads, analyze the ads for retrieving personal information and use this information to reconstruct user profile.
- Evaluated the system with multiple users, on multiple devices, captured over 5000 ads and analyzed the ads.
- Proved that if a user is using his or her mobile at office environment with restricted network and using the same device at home environment with an unrestricted network, following two different usage patterns we can still identify him or her to be the same user.
- The developed system can be used as one of the approaches in digital forensic readiness framework for mobile devices.
- This system and method have a wide range of applications in context-based security, proactive and reactive digital forensic investigation.
- The developed system can be used to identify any ads violating regional compliance.

E. Proposed a system and method for detecting vulnerabilities and predicting automated solutions to individual application binaries.

- An approach for creating a new wrapper to protect the existing apps, irrespective of their features.
- Reduced Software Development Life Cycle (SDLC) time for the app development, while reducing the risks of cyber-attacks.
- Internally wrapping the apps in a secure envelope to translate them into a new digital binary app without modifying their original functionalities.
- Keeping the app features intact, while the shielding process is carried out with well-defined protocols with no code variations to provide a relatively faster time to market.
- Introduced a machine learning system to suggest the best solution for the identified vulnerabilities, thus increasing the efficiency of the auto-fixing.
- Demonstrated a working prototype by evaluating the system with 5121 mobile apps, for recommending a security solution and perform auto fixing.

1.6 Thesis organization

The authors have proposed and implemented digital forensic readiness system for mobile computing systems, refer to figure 1.1. The contents of the thesis are organized in the following way:

- Chapter 2 provides an overview of digital forensic readiness (DFR), forensic readiness in mobile computing systems, mobile computing systems security landscape, the current research gaps and challenges that have motivated the present thesis work.
- Chapter 3 presents authors proposed design and implementation of precognition system, the security analysis and digital forensic readiness system targeted at smartphones, smart-watches, and wearables in an enterprise environment
- Chapter 4 presents a solution (iSecureRing) to secure mobile apps and to preserve date and time stamps of events to handle security incidents in the iOS operating systems. Forensic ready mobile app security solution to secure an application running in an insecure environment within the enterprise environment.
- Chapter 5 presents a forensic readiness solution for Android operating systems. Proposed and implemented an approach which gathers kernel generated timestamps of events and stores them in a secure location outside the device. In the case of a security incident, these stored timestamps can help in the offline digital forensic investigation.
- Chapter 6, the author proposes to extract ads on mobile devices, retrieve user-specific information to reconstruct the user profile and to predict the user's identity. The proposed system can be used as one of the techniques in digital forensic readiness framework for mobile devices. The system has a wide range of applications in context-based security, proactive and reactive digital forensic investigation.
- Chapter 7, the author has proposed a system and method for detecting vulnerabilities and proposing an automated fix to protect particular application binaries. The authors have used machine learning to recommend the most comprehensive solution.
- Chapter 8, the author summarizes the results presented in this thesis and suggest possible directions for future research.

Chapter 2

BACKGROUND

In this chapter, the author discusses the history of forensic readiness enabled systems, forensic readiness and current security landscape in mobile computing. Further, the author highlights the current research gaps and challenges that have motivated this thesis.

2.1 A brief history of Forensic Readiness enabled systems

While building secure systems, often the capability of the system to handle security incidents is ignored. Digital Forensic Readiness proposes to have process and procedures in place to collect digital evidence ahead of time. This help the enterprises in their preparedness to conduct a digital forensic investigation in case of any security breaches. It is essential that the systems be designed with forensic readiness objectives. Figure 2.1 provides the history of forensic readiness till 2014 and the figure 2.2 provides the history of forensic readiness from 2015-2018.

Rowlingson [116] proposes a ten step process for implementing digital forensic readiness by organizations. Grobler et al., [55] propose DFR as a best practice component for information security. Endicott-Popovsky et al., [40] discuss how cyber-attacks target networks to disrupt the services. In their work, the authors propose Network forensic readiness as an excellent measure to implement in enterprises. Mouton et al., [95] have proposed DFR for the wireless network and implemented it as an additional layer of protection.

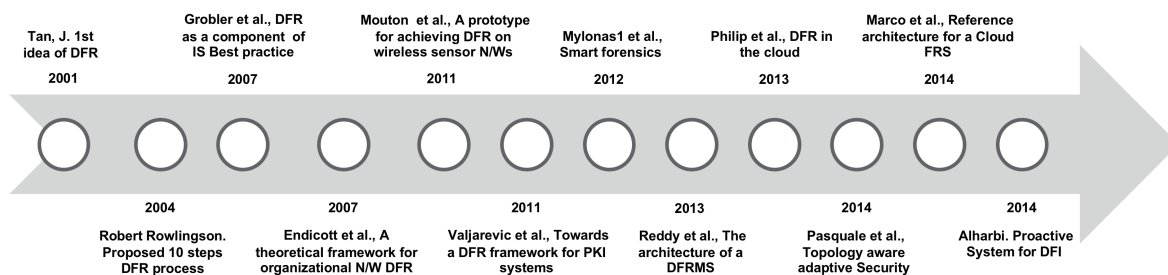


Figure 2.1: Forensic readiness time-line till 2014

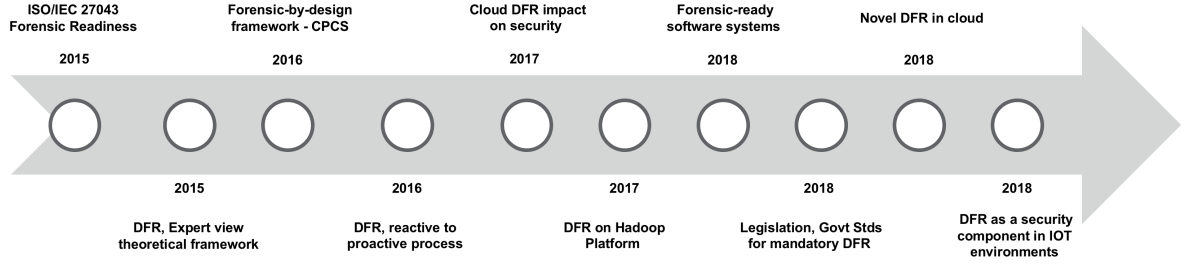


Figure 2.2: Forensic readiness time-line 2015-2018

Valjarevic et al., [138] proposed and experimented with a DFR for public key infrastructure (PKI) system. They took into account DFR should enhance the security and at the same time not altering the processes of the PKI system. Reddy et al., [114] have proposed DFR management system for large enterprises. Ruan, Keyun, et al. [117] analyzed DFR for cloud computing and implemented few prototypes. Dykstra et al. [36] evaluated some of the existing tools like EnCase in the context of Cloud forensic readiness (CFR). The results showed that the current tools were not reliable for CFR.

Dykstra et al. [37] proposed a remote forensic acquisition suite of tools. The suite, called FROST, provides a forensic capability at the IaaS level of OpenStack; an open-source Cloud Computing platform. FROST performs data collection from the Cloud service providers (CSPs) and the host operating system. The collected data includes virtual machines images, logs harvested from the API requests, and the OpenStack firewall logs. This suite is also considered as a way for adapting Forensic Readiness to the Cloud because it performs the necessary data collection activities.

Trenwith, P.M. and Venter, H.S. [137] have presented an approach for achieving Digital Forensic Readiness in the Cloud. It comprehends a remote and central logging facility for accelerating the acquisition of data; the model was also prototyped for windows platforms.

Pasquale, Liliana, et al. [111] proposes to monitor the changing network conditions within the network topology concerning adaptive forensics, security, and privacy. The authors aim to detect potential attacks by observing changes to the external operating conditions. Knowing possible attacks would help in targeted evidence collection rather than collecting evidence from all sources.

De Marco et al., [30] presented cloud forensics readiness reference architecture. The architecture proposes a forensic database to collect monitored data, services artifacts, and forensics logs. This data is used by the forensic readiness module to generate digital evidence, preserve digital

evidence, events time which is input to the incident response system and law enforcement communications.

Soltan [7] in his Ph.D. thesis talks about a system for Proactive digital forensics investigation. The reliable, proactive system was modeled as a dynamic feedback system in which the forward system is the one which is being investigated, and the feedback system is the proactive component. An event-driven model called digital forensic rules was used as the engine for providing input to the proactive component.

ISO/IEC 27043: 2015 International Standard, [1] describes the readiness processes that could be adopted by enterprises for their digital information system. The standard describes the processes for of evidence identification, planning pre-incident gathering/analysis, planning incident detection, readiness architecture, implementing, assessment and employing the assessment results.

Elyas, Mohamed, et al. [38] in their work review the existing literature and highlight the lack of guidance for implementing DFR. In their work, authors further validate and refine the DFR framework through expert group recommendations and also discuss issues facing practical implementation of DFR systems.

Ab Rahman, Nurul Hidayah, et al. [2] discuss cyber attacks on cyber-physical cloud systems (CPCS) and the need for designing CPCS which is forensic ready. The authors propose forensic by design for CPCS, suggest six factors for implementing the DFR framework.

Sachowski, Jason. [119] in his book discusses implementation guidelines for organizations to enhance its proactive DFR, discuss people, process and technology for effective DFR.

Alenezi, Ahmed, et al. [6] discuss the impact of forensic readiness on the security of cloud computing systems. Often in case of an attack, security and forensics are considered as separate tracks. This paper highlights the need for considering forensic readiness and security together to enhance the overall security of the cloud computing systems. This would help in enhancing the forensic capability of the organizations and ensure they are well prepared to handle any cyber attack.

Oo, Myat Nandar, and Thandar Thein [102] propose DFR on the Hadoop platform, the authors first identify the forensically important artifacts in the Hadoop platform (Non-Ambari Hortonworks Data Platform (HDP)) and propose DFR by analyzing these artifacts. The authors contribute towards effective evidence generation in the real-world Hadoop platform.

Pasquale, Liliana, et al. [110] discuss security requirements for creating forensic readiness based software systems. Analyze how current software development methodologies are not well-suited for developing such systems.

Park, Sungmi, et al. [109] discuss how the security breaches are because of poor implementation of security controls in private and government organization. Most of the times, security measure have been only a reactive response and did not address the threats. The authors survey the current data protection legislation of selected countries and initiatives taken to implement DFR. The authors recommend implementing DFR as a mandatory requirement for private and government organizations.

Kebande et al., [74] discuss implementation of DFR for cloud computing systems. The authors employ non-malicious botnet to function as an agent-based solution in the cloud environment. This technique help in performing DFR without disrupting the cloud environment operations. The DFR system align with ISO/IEC 27043: 2015 International standard.

Kebande et al., [75] propose requirements and processes required to implement DFR for secure IOT systems. The authors claim that the defined system complies with ISO/IEC 27043: 2015 International standard.

2.1.1 Forensic readiness for mobile computing systems

Yadav D et al.,[148] discuss mobile forensics to be more difficult and complicated considering the full range of hardware devices and software. At the rate at which mobile devices are being introduced into the market, every day there is a new device with new technology, new hardware, and new software. In their work author proposes mobile forensics approaches and highlights the challenges with the current strategy.

Mylonas et al., [96] have focused on ad hoc acquisition of smartphone evidence. The proactive smartphone forensic investigation scheme consists of three parties, investigator, independent authority, and the suspect. Here independent authority is the vital entity, one who controls the entire process of evidence collection, storage, and transmission.

From the literature survey, authors comprehended that there is a need for well defined forensic readiness framework for mobile computing devices at the application and operating system layer. In Chapter 3 authors have addressed this gap, by proposing and implementing the automated Digital Forensic Readiness System for Mobile Computing Devices targeted at smartphones, smartwatches, and wearables in an enterprise environment. In Chapter 4 authors have proposed

and implemented forensic readiness solution for iOS operating system and in chapter 5 authors have implemented forensic readiness for Android operating systems

2.2 Mobile computing systems: security landscape

Mobile devices have now become a more significant threat compared to the desktops and bring risks of security in the form of faults, errors, failures, and attacks; there is need to understand these risks to build secure and dependable mobile applications. There is a combination of applications that are being targeted for mobile devices. One set of developers are focused on migrating existing web applications for mobile devices. The second set of developers are focused on building new applications leveraging feature intensive devices using native development framework.

Developing secure mobile applications has become a responsibility of the development team. Mobile operating platforms only provide security guidelines. Often the team focuses on usability and functionality, and security is given least priority. Majority of application developers lack knowledge about security guidelines, security architecture and security threats. Frequently application functionality and secure coding are implemented together as one unit which makes it difficult to get the right balance between usability, functionality, and security. This process makes reuse of the security implementation challenging and leads to duplication of code [80].

2.2.1 Mobile application security threats

Consequently, along with the common threats that we comprehended in the desktop world, there are the new set of threats in the mobile world. There is a need to understand these threats and address them adequately. With widening mobile application security threat landscape, the applications are exposed to an increased attack surface. As per OWASP [76] [103] following are the Top 10 Mobile Application Security Threats. Refer to Table 2.1.

Table 2.1: OWASP Mobile Top 10 Threats

ID	OWASP 2014	OWSAP 2016
M1	<i>Weak Server Side Controls:</i> Poor web services hardening, insecure web server configurations, authentication flaws, session Management flaws, access control vulnerabilities and injection	<i>Improper Platform Usage:</i> Misuse of platform features or failure to employ security control
M2	<i>Insecure Data Storage:</i> Storing PII information in clear text [89]	<i>Insecure Data Storage:</i> Combination of M2 + M4 from 2014
M3	<i>Insufficient Transport Layer Protection:</i> Lack of certificate inspection, weak handshake negotiation and privacy information leakage	<i>Insecure communication:</i> Similar to 2014. Weak handshake, insecure SSL, communication in cleartext
M4	<i>Unintended Data Leakage:</i> URL caching, keyboard press caching, copy/paste buffer caching	<i>Insecure Authentication:</i> Failure to identify user, maintain user identity, weak session management
M5	<i>Poor Authorization and Authentication:</i> Non-validation of authenticated users, non-verification of authorized users and weak passwords/passcodes	<i>Insufficient Cryptography:</i> Incorrectly applied cryptography
M6	<i>Broken Cryptography:</i> Poor key management, using custom encryption protocols and usage of insecure or obsolete cryptographic algorithms	<i>Insecure Authorization:</i> Failure to authorize users
M7	<i>Client Side Injection:</i> SQL injection, XSS, local file injection, XML injection and format string injection	<i>Client code quality:</i> In secure coding on mobile device (Buffer overflow, format string vulnerability)
M8	<i>Security Decisions Via Untrusted Inputs:</i> Insecure Inter Process Communication (IPC) mechanism	<i>Code Tampering:</i> Hooking, method swizzling, dynamic memory modification
M9	<i>Improper Session Handling:</i> Failure to invalidate sessions, weak timeout protection, improper cookie management and insecure token generation	<i>Reverse Engineering:</i> Reverse engineering app binary to get the source code and exploiting from the inner working knowledge of the app. Same as M10 from 2014
M10	<i>Lack of Binary Protections:</i> Code-decrypting and reverse engineering	<i>Extraneous Functionality:</i> Hidden backdoor functionality, test stub code not intended for production release

There is the need for designing secure mobile systems that could detect such attacks and protect itself. In Chapter 4 authors present (SecureRing) solution to secure mobile apps in the jailbroken iPhones. Forensic ready mobile app security solution to secure an application running in an insecure environment within the enterprise environment. In Chapter 7 authors have implemented MobiSecureWrap solution, which detects security vulnerabilities, create Trusted Dynamic Library (TDL) solution and inject the application binary with TDL providing an additional layer of protection

2.2.2 Mobile application security analysis: tools and techniques

The following section describes the existing tools and techniques for performing security analysis:

ComDroid tool [24] analyzes the control flow of procedures and identifies inter-app communication vulnerabilities. Geneiatakis et al., [49] in their work, authors combine the outcome of static and dynamic analysis & compare with manifest's permission list identifying whether the Android application has overprivileged permissions or not. Li et al., [83] propose creating a call flow graph and thereby observing the data flow from source to sink to detect data leakage.

SCANDROID [44], performs data flow, pointer, and control flow analysis to validate if it complies with the security specifications and certifies whether the application is secure. Suzanna [133] analyzes various static analysis techniques for Android applications based on Confidentiality, Integrity, and Availability. Enck et al., [39] decompile the Android apps to recover Java source, performs static code analysis using Fortify SCA tool to identify security issues. Stowaway, [43] a tool to determine API calls and map them to the permissions for finding overprivileged permissions in Android applications.

Droid test [118], a server-side black box test tool which examines the inputs and output by running a set of test cases and correlated test cases and identifying if there any data leakage. Droid watch, [57] a prototype enterprise monitoring system for the Android apps for continuously collecting data of interest and perform analytics using tools like Splunk. Guido et al., [59] propose identifying mobile malware on enterprise devices, based on the changes occurring to device partitions, extracting only changed blocks of data and reconstructing images from them.

Shabtai et al., [124] propose applying machine learning techniques to features set for Android applications to classify whether the application is malware or not. Hasan, Basel, et al. [63] in their work, authors talk about security threats, what are countermeasure that mobile devices can adapt and highlight how user acceptance rate varies based on implemented security measure. This work helps in balancing security, functionality, and usage.

Sadeghi, Alireza, et al. [120] in their work presents the results of the literature survey performed in the area of security assessment of Android applications. The authors have analyzed over 100 papers published between 2008 to 2015. Their study reveals that the most of existing work is focused on the single app assessment and there is a need for more board and extensive analysis considering multiple applications from the similar domain, authors also suggest performing hybrid analysis for more comprehensive security analysis.

Li, Li, et al. [84] in their work present the literature survey for static analysis of Android apps.

The authors consider 124 papers published between 2011 to 2015. The authors reveal following key findings: most analyses are performed to identify security flaws in the Android app, the analysis is based on the single app analysis, taint analysis is widely used technique, existing approaches have missed considering at least one feature of android programming.

Titonis, Theodora H et al., [135] from Veracode inc in their patent, propose automated static and dynamic analysis of mobile apps. The technique involves users submitting their applications for analysis. The system analyzes the runtime behavior of the applications in a sandboxed environment. Analysis data collected for this app and learnings from past such executions is used to detect anomalous and malicious behavior to classify the apps.

Pistoia, Marco et al., [112] in their work propose Phoenix, a solution for combining the static analysis and machine learning to identify apps exhibiting malicious behavior.

Dolby, Julian T., et al. [34] in their patent propose installing an agent on the user's device, the agent then checks for newly installed apps, analyze for any possible security threat and generate alert to the user.

Xu, Zhi, et al., [147] in their patent describe performing static analysis of mobile applications followed by performing dynamic analysis based on the inputs from static analysis. Based on the dynamic analysis the application is classified to be whether it is malicious or not.

Wei, Fengguo, et al., [142] in their work, authors propose performing data flow and data dependence analysis in a component and track inter-component activities. This information is further utilized to perform inter-app and intra-app analysis. This technique help in identifying any security problems from the interaction between various components in the same app or different apps.

Wang, Wei, et al. [140] in their research, the authors propose employing an ensemble of multiple classifiers to detect whether an android app is malicious or benign. Authors extract about 11 types of static features from each app for dataset preparation. The authors have worked with large data set and were able to identify malicious app with an accuracy of 99.39% and detect benign apps with an accuracy of 82.93%.

Cimitile, Aniello, et al. [25] have proposed identifying malicious iOS applications employing machine learning. The authors extract the features from the static analysis of iOS applications. The authors were able to identify trusted applications with a precision of 0.971 and recall equal to 1.

Most of the current literature is focused on Android and not much work on automating iOS application security analysis, current solutions have automated static analysis and not on dynamic analysis. Current machine learning solutions have focused on classifying whether an app is malicious or benign using features extracted from static analysis, existing solutions have not leveraged machine learning techniques to improve the accuracy of vulnerability prediction, detecting one or more vulnerabilities, proposing best fix solution for identified vulnerabilities, and automatically fixing them. In Chapter 3 authors have proposed and implemented automated application security analysis on devices running Android/AndroidWear/iOS/WatchOS to detect potential security threats. In Chapter 7 the authors have proposed a system and method for predicting automated solutions and auto-fixing based on the detected vulnerabilities to protect particular application binaries. The authors have applied machine learning techniques for vulnerability detection and solution prediction to make it a learning system.

2.2.3 Handling security incidents

Forensic investigation [116] is employed after the event in response to a security incident. Forensic investigation is as good as the ability to collect relevant data. For a successful incident handling it is necessary to plan and implement the effective approach in place to anticipate such events and put systems in place to able to collect relevant data. Pangalos G et al. [107] describes the corporate forensic readiness in the information security framework.

Handling security incident has never been an easy task on mobile devices, considering the mobile platforms are getting more and more integrated with cloud computing, handling any security incidents has become all the more complex. [54] Security incident handling is one of the critical components of information security. With newer and advanced technologies security-related threats have become not only numerous but also diverse. With emerging new technologies, newer types of security incidents emerge frequently.

There is a need for proactive digital forensics which can help in handling security incidents swiftly. In Chapter 4 authors present a solution to preserve date and time stamps of events for the iOS operating system and in Chapter 5 for Android operating system to handle security incidents

2.2.4 Security and privacy threats through mobile ads

There are many others ways by which security of mobile applications is compromised. With smartphones, users can now view the ads on their mobile devices. Smartphones have opened up new ways of displaying ads to the users. User's browsing behavior is tracked by the search providers to construct their behavior profile, which according to them helps in fine-tuning user's search results and projecting user-specific advertising. Castelluccia [22] describes how in tar-

geted ads, user's behavior is tracked over a period to present the best possible ads for them; this poses potential security and privacy risk. User's behavior is exposed not only to targeted ads but any entity who have access to the ads. Based on harvesting information leaked from website ads, the author demonstrated that user's profile could be accurately reconstructed. With only a small number of ads websites links from Google ads, the system can infer 79% of user's interest and reconstruct up to 58% of her Google ad profile.

Ads seem to leak quite a bit of information about the user behavior. When a user search input matches with some commercial product or service offering, the search providers display ads based on the user's previously saved interests, likes, and dislikes. All of these user preferences are stored on the server side of search providers that are inaccessible to the user's machine. Extraction of such information without the help of search providers has several applications in digital forensics. In case of a security incident, this information could assist in gathering potential pieces of evidence from a given digital device.

Book, Tet et al. [18] conduct an empirical study which shows that nearly all the mobile ads are based on application and time base targetting, while 43% ads were based on location and 39% based on the user. This proves that information like application type, time, location and user information can be inferred.

Suman [98] created a tool MAdScope for characterizing the mobile In-App target Ads. The objective of the tool was to harvest the ads, probe ad network to characterize its targeting mechanism and emulate user profile to interact with the ad-eco system to understand if ad network does behavior targetting. The author concluded that location information is widely used by ad networks to target users.

Barford, Paul, et al. [13] Adscape conduct a study that seeks to understand the features, mechanisms, and dynamics of displaying ads on the web. Based on the study they report that when targetting ads are used, it generally based on details of user profiles and user's pattern of visits.

In Chapter 6 authors propose a novel forensic analysis approach to extract ads on mobile devices, retrieve user-specific information to reconstruct the user profile and to predict the user's identity.

2.3 Summary

There has been considerable research in the area of digital forensic readiness, researchers and practitioners have proposed several DFR approaches targeted at various IT systems. With the rise of mobile computing devices, need for DFR system targeted at smartphones, smartwatches and wearables emerged as an area that needs to be researched. Further, from the literature

survey, the author observed security of mobile devices, incident handling and vulnerabilities resolution as a critical concern. While there are many tools and solutions, applying machine learning to create a learning model is the gap that required to be addressed. A solution for mobile computing systems which are both forensic ready and secure was not well defined. These gaps and observations motivated the author towards this thesis. In next chapter 3, the author starts by proposing and implementing security analysis and digital forensic readiness solution for mobile devices in an enterprise environment.

Chapter 3

AUTOMATED DIGITAL FORENSIC READINESS SYSTEM FOR MOBILE COMPUTING DEVICES IN ENTERPRISES

In this chapter, the authors have designed a security analysis and digital forensic readiness system targeted at smartphones, smartwatches, and wearables in an enterprise environment. This work was presented at Annual ADFSL Conference on Digital Forensics, Security and Law in January 2018 [52]. The proposed system detects applications violating security policies, analyzes Android and iOS applications to identify possible vulnerabilities on the server, apply machine learning algorithms to improve the efficiency and accuracy of vulnerability prediction. The system continuously learns from past incidents, proactively collect required information from the devices which can help in digital forensics. Machine learning techniques are applied to the set of features extracted from the decompiled mobile applications and applications classified based on consisting of one or more vulnerabilities. The system was evaluated in a real-world enterprise environment with 14151 mobile applications and vulnerabilities was classified with an accuracy of 94.2%. The system can also work on virtual instances of mobile devices.

3.1 Introduction

Around 54% of the world population is using the Internet as per the Internet World Stats report 2017 [56]. On-line social networks (OSN) take the most significant share of Internet users, where about 2.77 billion users were active on social networks in 2017 [131]. Out of the digital equipment that people use to connect, mobile devices have the largest user base worldwide. The global population uses mobile applications for a variety of activities like social networking, online shopping, mobile banking, and for storing personal as well as official data. The Google

play store recorded about 94 billion app downloads [15] whereas Apple App store registered over 180 billion downloads [130] in 2017. The numbers are expected to further increase as indicated by one of the Gartner's reports that have predicted the total connected devices to touch 20.4 billion by 2020 [48].

The statistics are not different in the enterprises around the world. As per another Gartner's report, by the end of 2017 around 50% of employers would adopt BYOD (Bring Your Own Devices) mainly constituting mobile devices [46]. The increasing adoption of smartphones and wearables within enterprises would bring in additional attack vectors. Hackers and cyber-terrorists could target the vulnerabilities of mobile applications running on such BYOD devices to gain access to the respective organizations. The vulnerabilities could also be exploited for launching attacks, stealing data, bringing down organizational services to carry out other acts of hacktivism. The BYOD adoption in enterprises would also elevate security risks which arise from the Insider threats. As per Checkpoint's survey report in 2014 [31], around 82% of organizations and individuals observed a rise in mobile security incidents which is expected to increase every year.

There is a need for an intelligence-driven system, which can perform proactive security analysis and be forensic ready. The need has led to the proposal and development of a machine learning based system which the authors have named as '**Precognition**'. The proposed system combines both a proactive security analysis as well as forensic readiness into one solution that protects all mobile computing devices within respective enterprises. The authors would like to highlight following contributions of the Precognition system:

1. The system can monitor mobile apps installed on a device, or an equivalent virtual instance, in accordance with the enterprise defined security policies.
2. The system can automate the security analysis of the Android as well as iOS mobile application packages to identify potential vulnerabilities.
3. The system uses Machine learning for classifying such vulnerabilities to increase the efficiency and accuracy of the solution.
4. The system is digital forensic ready, i.e. it collects data required for digital investigations. If an attack happens that exploits a particular vulnerability, the forensic investigators would have quick access to potential pieces of evidence against the attacker.
5. The system was evaluated in a large enterprise in a live environment. The authors have collected and analyzed 14151 mobile apps.
6. The system provides insights based on industry verticals, app categories, and vulnerabilities distribution. These insights could be beneficial for the enterprises and application owners.

The rest of this chapter is organized as follows: Section 3.2 describes the Precognition solution architecture, Section 3.3 describes the implementation, Section 3.4 provides the results and inferences and Section 3.5 wrap-ups the chapter summary.

3.2 Precognition solution architecture

Precognition system has been designed to work on Android smartphones, Android wear, iOS smartphones, and iWatch. The Precog Client App is installed and run on the devices. The Precog Server processing components and databases are hosted on the enterprise application server or on-premise Cloud. Precog client App monitors the apps installed on the mobile devices and wearables as per security policies defined by an enterprise. The app-package-files and other forensically relevant logs are transferred to the local server, for the apps that violate security policies.

In the next step precognition server component running on the server identifies all potential security threats by running both offline-analysis as well as the online-analysis. The offline-analysis is performed by reverse engineering of the application package, whereas the online-analysis is performed by analyzing the app running on the device or an emulator. After the completion of offline-analysis / online-analysis, a list of issues that help in identifying the vulnerabilities is obtained.

The authors have trained a machine learning model to classify vulnerabilities based on the list of issues identified in the prior step. The trained model helps in identifying the unforeseen malicious apps running on mobile devices and wearables. The identification of such malicious apps helps in ensuring the forensic readiness of the system. In the case of any security incident, the investigator can use the collected information for conducting a forensic investigation. Figure 3.1, shows the proposed architecture. The solution architecture consists of six main components:

1. *Mobile computing devices (BYOD environment)* - All the devices are configured to run two instances: a personal and a corporate. The instance could be a virtual image or a sandboxed profile as supported by host operating system. The current solution focuses on the corporate instance on which the Precog Client app is installed. In case of Enterprises that do not enforce the division of profiles, the Precog Client app can be directly installed on the device. The app monitors the device as per the enterprise defined security policies as follows:
 - i All Apps.
 - ii Apps downloaded from the untrusted marketplaces (Any app not downloaded from the whitelisted app stores. Currently, the solution checks for the Apple App Store,

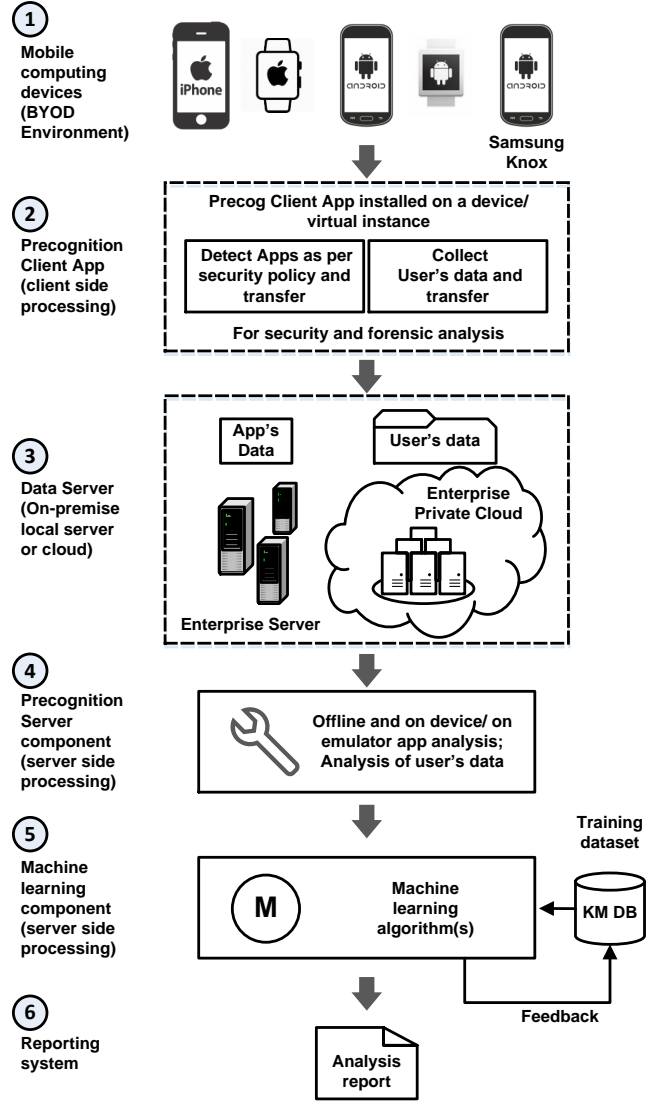


Figure 3.1: Precognition system solution architecture.

the Google Play Store, the Amazon App store, and the Samsung app store).

- iii Apps which need access to special privileges. Currently, the solution checks for apps which require access to SMS, Camera, photo gallery, and emails.
- iv Virtual instance backups.
- v The backup of selected apps user's data.

The first option of monitoring all apps may not be practical for large enterprises, considering the amount of processing required for analysis. However, this security policy can be enforced for a selected group of high-risk individuals (like CEO, COO, and other Senior Management personnel) within the enterprises. The second option of monitoring apps downloaded from untrusted sources is a more practical approach. However, there is a risk

of overlooking malicious apps from whitelisted sources. In the real world scenario, the enterprises can setup security policies by adding new, modifying or deleting the existing policies. After the setup, they can configure a combination of various security policies based on the requirements. In the current prototype, the authors have implemented the second option checking for apps from untrusted sources.

2. *Precognition client App (Client side processing)* - All devices have the Precog client app preinstalled, which filters apps according to the enterprise's security policy, collects app-specific data, collect user's data and transferred to the server for further processing
3. *Data Server* - Stores app and user data for further processing. The app data consists of app-package files (.apk and .ipa), whereas, the user data includes data generated by the user like call records, SMS, camera photos/videos, audio files, browsing history and contacts.
4. *Precognition Server Component (Server side processing)* - Includes Offline and On Device/Emulator Security Analysis of untrusted apps to identify security vulnerabilities. The selected user's data is utilized for security and forensic investigation.
5. *Machine Learning Component (Server side processing)* - Includes a trained learning model, a knowledge database and continuous feedback to the model. The Machine learning algorithms aid to classify vulnerabilities and improve the accuracy and efficiency of the system.
6. *Reporting system* - produces security and forensic analysis reports, that recognize the threat before it can happen, so that the security incident can be prevented.

3.2.1 Precognition work-flow

The following section explains the Precognition System Client server workflow.

Client side workflow- the Precognition client app detects apps as per security policy and transfers them to the Server for further analysis. The client app takes the backup of the user app data in case a virtual instance needs to be checked. This backup is stored in the enterprise cloud.

Server side workflow- the Server component runs on the enterprise server, where it is responsible for:

- *App Analysis* - it picks up the uploaded apps and performs the security analysis. Afterward, it uses machine learning to identify all possible security vulnerabilities and threats.
- *Virtual instance analysis* - In case the check needs to be done for a virtual instance running on a device (like the work-instance on Samsung Knox), the server analyses the app data and the corresponding user's data of the reported app. The analysis happens as per the security policy and follows the same process as done for a standalone device.

Table 3.1: Forensic and security analysis of apps

Forensic / security analysis	Android and Android-Watch apps (.apk)	iOS and watchOS apps (.ipa)
Detecting untrusted app	GetInstalledPackages[], Validate based on the app-package name	GetInstalledappbundles[], check if the file embedded.mobileprovision exists
Permission analysis	adb commands to get respective apk-information, get permissions list, permissions analyzed	
Reverse engineering	Get .smali files, java classes, and manifest files	Plist, Resource files, Header files, assembly code
Logcat analysis	Get logcat information	
Run exported activities	Fetches the ExportedActivityNames	
Insecure data storage analysis	The text or the cache or the database files	Plist and the SQLite databases
Unintended data leakage analysis		Cookies, caches, and the snapshots, snapshots are copied and displayed to the user for analysis manual analysis.

Table 3.2: List of devices utilized

Operating System	Device
Android	Samsung Galaxy S3
Android	Samsung Galaxy S7 with Knox
Android Wear	Asus ZenWatch
iOS	iPhone 4, iPhone 5s
watchOS	Apple Watch Series 1

- *User's data analysis* - As part of the previous point, a forensic analysis of user data (SMS, Voice, Mail history, Call history) is carried out in the cloud.
- *Incident handling* - If an attack happens to exploit a particular vulnerability, the security analysis information will help in understanding how the attack was carried out. With the collected forensic information, the forensic investigators would be able to link the attack with the evidence against the attacker responsible for it.

3.3 Precognition prototype implementation

In this research work, the authors have implemented a working prototype of the proposed solution on Android and iOS devices (phones and wearables). The solution works on the security policy rule of scrutinizing all 'Apps downloaded from the untrusted marketplaces'. The table 3.2 enlists the devices that the authors have used in the implementation process. The table 3.1 compares the forensic and security analysis performed. All the files are scanned for sensitive keywords, like PII information, hard-coded values, and insecure APIs

3.3.1 Precognition client app

The Precog Client app for *Android* and *Android Wear* was created in Java; whereas, the authors have used Objective-C to write the app for *iOS* and *watchOS*. The Precog Client app is installed either on the device or the virtual instance. The authors also created a virtual instance test case using **Samsung Knox**, which is a feature available only on specific Samsung devices. When activated, Samsung Knox allows users to run two virtual instances on the phone; one for personal work and the other with a workspace environment dedicated to office work.

Detecting the untrusted .apk/.ipa

Android and Android-Watch apps (.apk) - The Precog Client app gets the list of all installed and newly downloaded apps on the device through the `GetInstalledPackages[]` command. The authors then validate if the packages belong to any of the following trusted sources based on the app-package name; namely, `com.android.vending` (for the Google Play), `com.amazon.venezia` (for the Amazon Appstore), and `com.sec.android.app.samsungapps` (for the Samsung Galaxy Apps - store).

If the `PackageName` does not belong to the Google Play store, the Amazon Appstore or the Samsung Galaxy Apps store, then the package is marked as suspected.

iOS and watchOS apps (.ipa) - The Precog client app gets the list of all installed apps using the `GetInstalledappbundles[]` command, and scans through the bundles to check if the file `embedded.mobileprovision` exists in the app resource path or not. If the file is present, then the authors consider that app as untrusted and mark it suspected.

In the current prototype implementation, the authors have executed all their tests on a jailbroken iPhone; because of the iOS's sandboxing feature that does not allow one app to access another app's information on any of its devices unless they are jailbroken.

Transferring the suspected apps to external storage system

The Precog Client App process through all suspected apps identified in the previous step 3.3.1. For the *Android devices*, it copies the suspected apps to a folder created on the memory card/ the external storage. Whereas, for the *iOS devices*, it copies all the suspected apps to a predefined private folder on the device, which is easily accessible on a jailbroken device. The copied apps are then compressed into a zip file.

Uploading the suspected/untrusted apps

All the suspected/untrusted apps are available in their respective folders after the previous step 3.3.1 concludes. The authors installed an Apache Server, which is managed by XAMPP app,

on the Precog Server where all the suspected/untrusted apps from the *Android devices* are uploaded.

In case of the *iOS devices*, all suspected/untrusted .ipa's are uploaded to the Precog Server which the authors installed using an Apache Server managed by XAMPP app on a separate Mac machine. The PHP web service is launched on the server to receive all these uploads.

Uploading user's data to the server

Android virtual instance - The Precog Client app uses tools like *Ultimate Backup Lite*, *Syncios*, and *Wondershare MobileGo* to take the backup of user's data (refer to the table 3.3 for user data categories). The un-rooted device backup contains `app/data/<package name>` which consist of only apk files, whereas, the rooted device backup contains `app/data/data/<package name>`.

A typical Android app contains the following data - Database/ : contains app's database, Lib/: holds libraries and help files, Files/: other related files, Shared_prefs/: all preferences and settings, and Cache/: keeps all caches. In case the device is rooted, SQLite files can be found under `app/data/data/<package name>/database/filename.db`, and the cache files can be found under `app/data/data/<package name>/cache/binary format`. The images can be found for both rooted as well as the un-rooted device under `photo/storage/sdcard0/`.

The data is be stored in XML format; for example, the `Contacts/callhistory.xml` looks like

```
<contact>
<displayname/>
<givenname/>
.....
<number>8105946466<number/>
<address>Address<address/>
<photo/>data will be in byte text<photo />
<email/>aaa@gmail.com<email/>
```

Table 3.3: User data backup

User Data	Files
Video	Storage/sdcard0/ all the video file
Audio	Storage/sdcard/ all the audio file
Photo	Storage/sdcard/all the photos
DCIMPhoto	Storage/sdcard/All images taken by device camera
App	data/app/all apk files
Call History	callhistory.xml Call details and duration of the call
SMS	sms.xml Sms in plain text
Contacts	contacts.xml contact numbers, names & address

3.3.2 Precognition server component

The server-side application provides the following two functionalities: carrying out the security analysis of app binary packages, using machine learning for identifying vulnerabilities; and arranging the security incident handling.

Android and Android Wear The server-side application is created using Java, with the following pre-requisite requirements: Java Runtime Environment (1.8 or above); Environment Variable "JAVA_HOME" set to the location of Java executables; Android SDK installed; Environment Variable "ANDROIDHOME" set to the location of Android Platform-tools; Windows Operating System (7 or above); R installed; Rooted Android devices with USB Debugging enabled in Developer Options; and Device driver from manufacturer for the concerned device installed on the system.

iOS and watchOS The server-side application is implemented using Objective-C/Java, with the following pre-requisite requirements: Java Runtime Environment (1.8 or above); Xcode and its command line tools installed; Eclipse IDE installed; a MAC System; and a jailbroken iOS devices with SSH, and SFTP protocols installed on it.

3.3.3 Server component: offline and on-device/emulator security analysis

The server component has following four main modules: 1. Inbox; 2. Polling service; 3. Precog offline module; and 4. Precog online module (refer to figure 3.2.)

Inbox - receives the untrusted .apk/ipa from the Client-side component.

Polling service module

Android and Android Wear This module is created as the windows service, which aims to poll the Inbox folder for the incoming untrusted .apk packages and then to hand them over to the Precog Parallel processing component. The Precog Parallel processing component is created using the *ThreadPool* class in C-Sharp to manage multiple threads. The numbers of threads are controlled by two parameters `ThreadPool.setMaxThreads` and `ThreadPool.setMinthreads`. In the current work, three apps can be processed simultaneously, and the 4th app will be in the queue until any of the threads become available. However, the maximum number of threads can be set based on available processing power. The untrusted app is then handed over to Precog Offline and Online processing components separately.

iOS and watchOS The module, which has been created as an Objective-C console application, polls the Inbox folder for incoming .zip files, and subsequently invokes a Java component (jar file) with the current .zip files' paths for offline and online security analysis. The polling

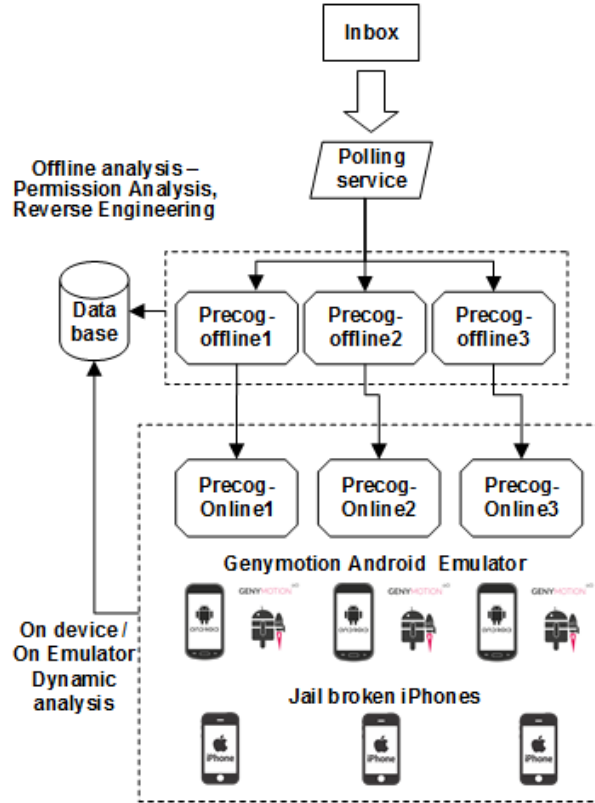


Figure 3.2: Server component

service is implemented with *NSOperation* and *NSOperationQueue* of Objective-C for parallel processing and queuing. The polling service will invoke maximum three Java components, i.e., analysis of 3 files can be run parallel and the next zip files uploaded are maintained in a queue

Precog offline processing component (.apk)

Precog offline processing component is a Java application packaged as Precog.jar file which takes the following parameters as inputs and performs .apk permission analysis and Reverse-engineering.

- .apk file's location.
- ClassKeywords.xml - contains a list of sensitive API names and the corresponding threat(s) due to usage of those API's.
- PIIInfo.xml - contains a list of personally identifiable information (PII) keywords [89]. i.e., Email, password, credit card number, country, zip code, full name, screen name, gender, telephone number, date of birth, driver's license number, birth place, IP address, login name, finger prints, vehicle registration plate number, digital Identity number, genetic information, passport number etc.

Android permission analysis The authors have analyzed 3,622 top rated apps, and created a database of permissions versus apps category for the same. For the selected app, the authors executed batch `adb` commands to get respective apk-information. The list of obtained permissions was compared against the database to identify all possible overprivileged permissions.

Reverse engineering The untrusted app is reverse engineered to its respective .smali files, java classes, and manifest files. The authors then run a batch execute the APKTool command to decompile the given .apk file to its corresponding .xml file and .smali file. Afterwards, the authors decompiled the .apk to its java classes using *Dex2jar* and *jad* decompilers. All the source files are then parsed against the previously discussed classkeywords.xml and PIIInfo.xml files. The identified security analysis information is stored in the database. The output is shown in the figure 3.3; includes the file-path, line number, sensitive information, function name, possible threat based on usage of that function, and the description of the threat.

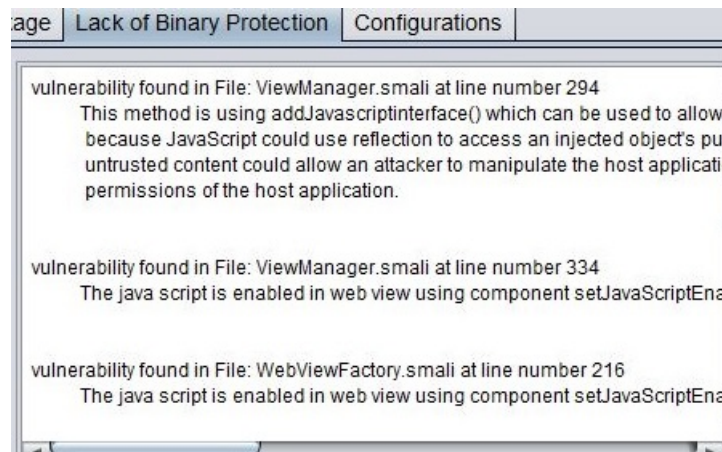


Figure 3.3: Lack of binary protection.

Precog offline processing component (.ipa)

Precog offline analysis consists of a Java component that invokes the Java services to perform the reverse engineering of the untrusted app. Unlike the .apk files which can be decompiled to get the corresponding source code in one go, the .ipa files need a **two-step process**. The .ipa package is first fed to a set of tools (Class-Dump) to obtain the corresponding **header files**; which are checked for flaws that result from insecure programming practices, like hard-coded values and use of inherently insecure APIs. The second step involves treatment of the .ipa package through a different set of tools (oTool) that produce the corresponding **assembly code**; which should be checked for human programming errors and malicious code bytes. Both the steps as mentioned above are required to get a complete analysis of given untrusted .ipa package; hence the authors have performed both whose details are presented in the following paragraphs.

Reverse engineering .ipa to header files The selected zipped app bundle is unzipped using a shell script to get its resource files and corresponding app binary. Then the app binary is reverse engineered by decompiling the app binary file to its corresponding header files (by using the **Class-Dump** command) [99]. These header files are then scanned for sensitive keywords, like PII information, hard-coded values, and insecure APIs. The PList and resource files, from the app bundle, are also scanned for such sensitive keywords. The program then parses through all these files to find out all potential security flaws related to the sensitive info, PII, or the insecure APIs. The information (file-path, sensitive information) is then stored in the database.

Reverse engineering .ipa to assembly The selected app is reverse engineered by calling a shell script to execute **oTool** [151] command to decompile the binary file to its corresponding assembly code. The parsing is performed throughout the complete assembly code to find out any security flaws related to sensitive info or PII, or any insecure APIs. The respective flaws are then flagged and displayed to the user. The identified information (file-path and sensitive information) is stored in the database.

Precog online processing component (.apk)

The untrusted .apk package is installed on **Genymotion** Android instance using **adb** commands to perform the dynamic analysis. In the current prototype, three Genymotion instances were set up to run three threads in parallel. The program takes two inputs - firstly the decompiled .apk folder path, and secondly the XML file containing the PII information to perform the following three types of analysis:

Logcat analysis Captures the logcat information and parses it for sensitive information disclosure, which is later stored in the database.

Run exported activities The program fetches the `ExportedActivityNames` from the manifest file. For each activity in `ExportedActivityNames[]`, a batch execution of `adb shell am -n` starts returning `packageName` and `activity` (takes a snapshot of the activity).

Insecure data storage analysis Traverses through the application folder to search for any sensitive information (or PII) in the text or the cache or the database files. The output of this step is shown in figure 3.4. Example:- `/file1/xyz.apk - 123 - Lastname`.

Precog online processing component (.ipa)

Since the authors designed the polling service to run three packages in parallel at a time, they used three jailbroken iPhones for performing the online analysis. The jailbroken iPhones were connected to a Mac machine, which was on the same network. The untrusted app is installed and

APK Info	Insecure Data Storage	Unintended Data Leakage	La
The database file "Chinook_Sqlite.sqlite" at file path C:\Users\Swapneil_Das h\AppData\Local\Temp\android_pt_temp\Chinook_Sqlite.sqlite might contain sensitive info : credit card			
The database file "Chinook_Sqlite.sqlite" at file path C:\Users\Swapneil_Das h\AppData\Local\Temp\android_pt_temp\Chinook_Sqlite.sqlite might contain sensitive info : Full Name			
The database file "Chinook_Sqlite.sqlite" at file path C:\Users\Swapneil_Das h\AppData\Local\Temp\android_pt_temp\Chinook_Sqlite.sqlite might contain sensitive info : Full Name			
The database file "Chinook_Sqlite.sqlite" at file path C:\Users\Swapneil_Das h\AppData\Local\Temp\android_pt_temp\Chinook_Sqlite.sqlite might contain sensitive info : Full Name			

Figure 3.4: Insecure data storage

```
IP Address '192.168.43.100' found in the application as a Hardcoded String.

IP Address '127.0.0.1' found in the application as a Hardcoded String.
```

Figure 3.5: iOS insecure data storage

launched on the jailbroken iPhones using SSH and SFTP protocols, and the Xcode commands from the Mac machine. The authors performed the following two types of analysis on the same:

Insecure data storage analysis The files like the plist and the SQLite databases from the app data folder are transferred to the Mac machine using SFTP commands. Afterwards, the Java component of online analysis is run and all these files are scanned for sensitive keywords, and PII information. The output is shown in the figure 3.5 is stored in the database.

Unintended data leakage analysis The files like cookies, caches, and the snapshots which are present in the app data folder are transferred to the Mac machine using SFTP commands. Afterwards, the Java component of online analysis is run, and all the files are scanned for sensitive keywords, PII information. The snapshots are copied and displayed to the user for manual analysis. The output is stored in the database.

3.3.4 Machine learning implementation

The machine learning model created to identify the vulnerabilities in the applications is based on authors previous findings and results of the analysis of such apps. The offline and the online app analysis reports are processed to build the training data, which is used to generate a learning model. Once built, the model can be used to predict the vulnerabilities for a new app. The training data is regularly updated with the predicted values to improve the accuracy of future

predictions. The approach consists of the following steps:

Training data - The past security analysis reports generated by the Precog solution are taken as the input for the creation of the training dataset. A set of keywords is used as the features for the training data. These features include any sensitive APIs and keywords which are likely to be present in an Android application. The training data consist of one record per app that were analyzed using the Precog solution.

Learning Model Generation - The authors have used the Naive Bayes algorithm (NBA) for classifying the apps based on whether the vulnerability is present or absent. We choose NBA for the following reasons [23]: training dataset is relatively smaller, consists of a large number of predictors, requirement for scalable architecture, need for the faster algorithm and also need for multi-classification. The algorithm calculates the probability for each of the categories of the vulnerabilities, and then assigns the vulnerability with the highest probability. One application can have more than one vulnerability, hence the authors choose *one vs. the rest* method for multi-label classification [81]. In multi-label classification, a model is generated for each of the vulnerabilities. For instance, if we have a vulnerability ‘Application Misconfiguration’, we use a binary classification approach in which we have only two classes: 0 and 1 where 0 denotes ‘Vulnerability Absent’ and 1 denotes ‘Vulnerability Present’. Similarly, a model is generated for each of the vulnerabilities. Once this process is complete, then the training model can be used to predict the vulnerabilities.

Feedback system - After predicting the vulnerabilities, the record, that contains the features and the predicted values, is updated into the training data. The updated dataset can be used for future predictions.

Predicting vulnerabilities - The test data is input to each of the models for predicting all the vulnerabilities. The machine learning implementation consists of three program modules. The first one is a Java program which is used for creating the Training data set. The second Java program is used to create Test data for the new app; and The third is a R program that is used for creating the learning model (refer to the machine learning flowchart shown in figure 3.6).

Training dataset creation (Java method to create training data)

Inputs: All the past security analysis reports from the database and a configuration file with all the keywords to be searched in the report.

Keywords:

ADB Backup, TelephonyManager.getdeviceID, setJavaScriptEnabled, getInstallerPackageName, logging some information or data within the code, PRAGMA key, Runtime.getRuntime.

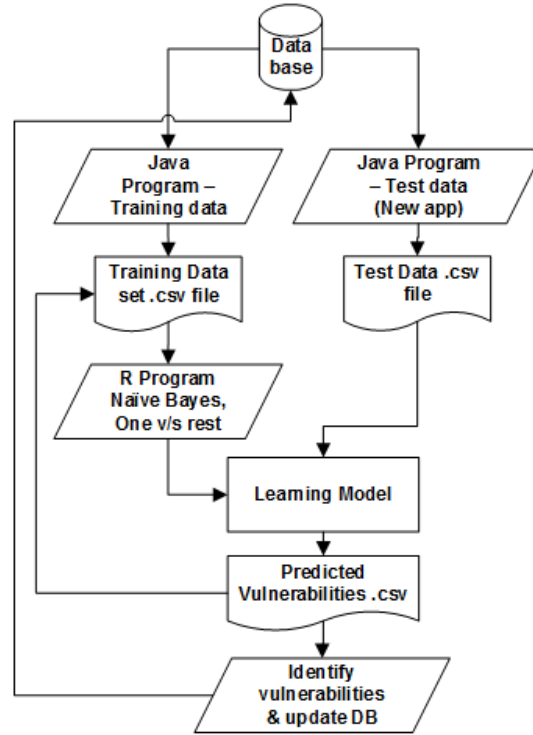


Figure 3.6: Machine learning flowchart

exec(...), getInstallerPackageName, addJavascriptinterface(), Keystore, Email, password, credit card, Cookies, Country, android:exported =true, Grades, Zip Code, Full Name, screen name, Gender, Telephone number, Date of birth, Driver's license number, Birthplace, IP address, Login name, fingerprints, Vehicle Registration plate number, Digital identity, Genetic information, Passport, Environment.getExternalStorageDirectory, sendDataMessage, sendMultipartTextMessage, sendTextMessage, setAllowFileAccess, Sqlite, get-InstallerPackageName, System.loadLibray, Runtime.getRuntime.exec(...), dexClassLoader(), debug mode is explicitly on, debug mode is explicitly off, sqlcipher, Settings.Secure.ANDROID_ID, setAllowFileAccess(false), MODE_WORLD_READABLE, sharedUserId, Secure. ANDROID_ID, google map.

Method: the Java program reads all the past security analysis reports generated by the Precog solution and generates the training data by parsing for each of the keywords which are stored in the configuration file. It would be run only once for creating an initial training data. If the keyword is present, '1' will be updated, else a '0' is updated. This process repeats for each application security report. For each of the apps, the vulnerabilities are assigned manually by analyzing the app for the present vulnerabilities. The OWASP Mobile top 10 vulnerabilities are considered for classification (refer to the table 3.5).

Output: The output is a TraningData.csv file that consists of a vulnerabilities list along with the set of keywords for each app.

Table 3.4: Training data set

	Vulnerabilities					Keywords				
AppName	M1	M2	M3	...	M10	P1	P2	P3	...	Pn
aLogCat	1	1	0	1	1	1	1	1	0	0
Amazon Shopping	0	1	1	0	1	1	0	0	0	0
Angry Birds	1	0	0	1	0	0	0	0	0	0
Asphalt	1	1	1	1	0	1	1	1	0	0
...					
Book-MyShow	1	0	0	0	0	0	1	0	0	0
BOX8	0	0	1	1	1	0	1	1	0	0
BusyBox	0	0	1	0	0	0	0	0	0	0
...					
Callapp caller ID	0	1	0	1	1	1	1	1	0	0
Calorie Counter	1	1	1	0	1	1	1	1	0	0
Cam Scanner	0	1	1	0	1	0	1	1	0	0
Canara Enfobook	0	0	1	1	0	0	0	0	0	0
Candy Crush	0	1	0	0	0	1	1	1	0	0
CirclePay	0	0	1	1	0	0	0	0	0	0
Clash of Clans	0	0	1	0	0	0	0	0	0	0
Cloud-Check-Service	1	1	0	0	0	1	0	0	0	0
...					
DB Summit	0	0	0	1	1	0	1	1	0	0
Dominos	1	0	1	1	1	1	0	0	0	0
Download Manager	0	1	0	0	0	1	0	1	0	0
...					
English Grammar	0	0	0	1	0	0	1	1	0	0
eRail	1	0	0	0	1	0	0	1	0	0
ESExplorer	0	0	1	1	1	1	0	0	1	0
...					
Facebook	0	1	0	0	0	0	1	1	0	0
Flipkart	1	1	1	1	1	1	0	1	0	0
FolderSync	1	0	1	1	1	1	0	0	1	1
Food Ordering app	1	1	1	0	1	1	1	0	0	1
...					
Gmail	1	1	0	0	1	0	1	1	0	0
...					
Zip-Recruiter	1	1	1	1	0	1	1	1	0	0

Test dataset creation (Java method to create test data)

Inputs: A configuration file with all the keywords to be searched in the report file and the string values from the database.

Method: The method reads the new app security analysis data from the database, then checks for each keyword (P1 to Pn) from the configuration file. If the keyword is present, a '1' otherwise a '0' is updated. The output is a TestData.csv file with a list of all the keywords for each app. It also has the list of vulnerabilities (M1 to M10) initially marked as 0 (refer to the table 7.3 for sample Test data set, and refer to the table 3.5 for vulnerabilities list).

Table 3.5: OWASP top 10 mobile vulnerabilities

Code	Vulnerability
M1	Weak Server Side Controls
M2	Insecure Data Storage
M3	Insufficient Transport Layer Protection
M4	Unintended Data Leakage
M5	Poor Authorization and Authentication
M6	Broken Cryptography
M7	Client Side Injection
M8	Security Decisions Via Untrusted Inputs
M9	Improper Session Handling
M10	Lack of Binary Protections

Table 3.6: Test data set

Vulnerabilities					Keywords				
M1	M2	...	M9	M10	P1	P2	P3	...	Pn
0	0	...	0	0	0	1	1	...	1

Java-R integration (Java method for calling the R program from Java)

The method first calls, then executes the R program from the Java program for the learning model generation.

The R program – The program takes the training data and test data as the input, then applies the Naive Bayes Algorithm to generate a model. The resulting model from the algorithm is used on the test data. The predicted values are stored in a file which is read by the Java program. The R program then updates the training data using the test data and the predicted values. The Naive Bayes algorithm uses the Bayes theorem for finding the probabilities:

$$p(Ck|x) = \frac{p(Ck)P(x|Ck)}{P(x)} \quad (3.1)$$

Where, $p(Ck)$ is the probability of a vulnerability being present in the complete dataset. $P(x|Ck)$ is the probability of a particular instance given its vulnerability. $P(x)$ is the probability of an instance given complete dataset. $P(Ck|x)$ is the probability that a particular vulnerability for a given app is present.

We find $P(Ck|x)$ for each of the vulnerabilities. The predicted vulnerability will be the one which has the highest probability. A simple implementation of Naive Bayes can do only multiclass classification; however, an app can have more than one vulnerability, so the authors employ *one vs. the rest* method for multi-category classification, where a model is generated for each of the vulnerabilities. For example: If we have a vulnerability ‘Application Misconfiguration’, we use a binary classification approach in which we have only two classes: 0 and 1 where 0 denotes ‘Vulnerability Absent’ and 1 denotes ‘Vulnerability Present’. Following a similar approach, the model is generated for each of the vulnerabilities. The output of this would have the predicted values for each of the vulnerabilities in a comma-separated file. If a vulnerability has a 0 value, then it is not present; otherwise, the vulnerabilities with a value 1 denote their presence.

Predicting vulnerabilities (Java method to read predicted data)

The method reads the predicted values from the file and maps it to the vulnerabilities, then displays the list of the predicted vulnerabilities.

Input: The file containing predicted values.

Method: The method reads the data from prediction file which has 0s and 1s for respective absence and presence of each vulnerability. If it is a 1, then the corresponding vulnerability is displayed.

Output: List of predicted vulnerabilities.

Feedback system (R program)

The R program which is used for predicting the vulnerabilities, after updating the vulnerabilities to the CSV file also updates the training data, forming a feedback system. Hence, the predictor values from the test data as well as the predicted values are updated to the training dataset. Thus, the training data is updated after the prediction of each app. This makes the system to be more accurate by continuously improving the learning model.

3.4 Results

Machine learning results: In order to measure the predictive ability of the proposed model, the authors have tested the accuracy on a set of data which is not used in the estimation process. The authors call the data as the “Test Set” and that used in the estimation as the “Training Set”. The authors have used the Mean Squared Error (MSE) to measure the proposed model’s predictive accuracy. The equation is as follows:

$$MSE = \frac{1}{n} \sum_{i=1}^n (y_i - \tilde{y}_i)^2 \quad (3.2)$$

where y_i - vector of i predictions, \tilde{y}_i - vector of observed values.

In the above situation, “leave one out cross-validation” is one of the approaches that can be utilized. There are two modifications of this technique, namely **leave-k-out cross-validation**; where k observations are left out at each stage, and **k-fold cross-validation**; where the original data is randomly divided into k sub-datasets, and one is left out in each iteration. In the current implementation, the authors have realized a **k-fold cross-validation** for testing the accuracy of the proposed model. To obtain the accuracy measures, the authors have considered n independent observations, $y_1, y_2, y_3, \dots, y_n$; where $k = 5$. The process is explained in the following steps:

1. Partition the dataset into k equal parts, say (k_1, k_2, k_3, k_4, k_5).

Table 3.7: Accuracy of vulnerabilities predicted

Vulnerability	Accuracy
Weak Server Side Controls	100%
Insecure Data Storage	93%
Insufficient Transport Layer Protection	86%
Unintended Data Leakage	93%
Poor Authorization and Authentication	85%
Broken Cryptography	100%
Client Side Injection	92%
Security Decisions Via Untrusted Inputs	100%
Improper Session Handling	100%
Lack of Binary Protections	93%
Average	94.2%

2. Let observation set k_i form the test set, and fit the model using the remaining data. Then compute the error e_i^* for the omitted observations. The result is called “predicted residual”.
3. Repeat step 1 and 2 for $i=1, 2, 3, \dots, k$.
4. Compute the MSE from $e_1^*, e_2^*, e_3^*, \dots, e_k^*$. The authors call it the CV.

The authors divided the training data into two sets, namely the training data set and the test data set. The training data set forms 80% of the total volume of the data, while the test data is the rest 20%. The variable “ k ” specifies the number of times the cross-validation approach is used. In each round, the authors took a different set of test data (20% of the total volume) such that no instance of data gets repeated for the following rounds. The no-repetition policy ensures that the algorithm’s accuracy gets tested on different kind of data in each iteration. The authors got a mean accuracy of 94.2%, which was obtained by considering the prediction-accuracy of all the vulnerabilities followed by taking the average of the same. The authors obtained Precision of 0.841, Recall of 0.833 and F1 score of 0.86. The authors have also included a feedback system that updates the training data which in turn increases the accuracy of the algorithm’s prediction. The table (3.7) presents the accuracy figures obtained while predicting the OWASP Top 10 vulnerabilities.

The authors have tested the system on 14,151 apps, which includes both Android (.apk) as well as iOS (.ipa) apps. Top Apps belonging to various industry vertical and application categories were downloaded for the test. The figure 3.7 shows the vulnerabilities distribution, where the word Domains refers to Industry verticals under which a particular app falls, and the word Category denotes the group name assigned to the particular app by its developers/owners. The authors observed that Shopping, Travel & Local, Business, Finance, Lifestyle are the top five categories which are most vulnerable to exploits. The figure 3.8 shows the distribution that depicts domain versus the types of vulnerabilities. The authors observed that the **Lack of binary protection**, **Insufficient transport layer protection**, and **Unintended data leakage** are the top three vulnerabilities across all the domains (refer to figure 3.10 for details. Moreover, **Retail**, **Media**

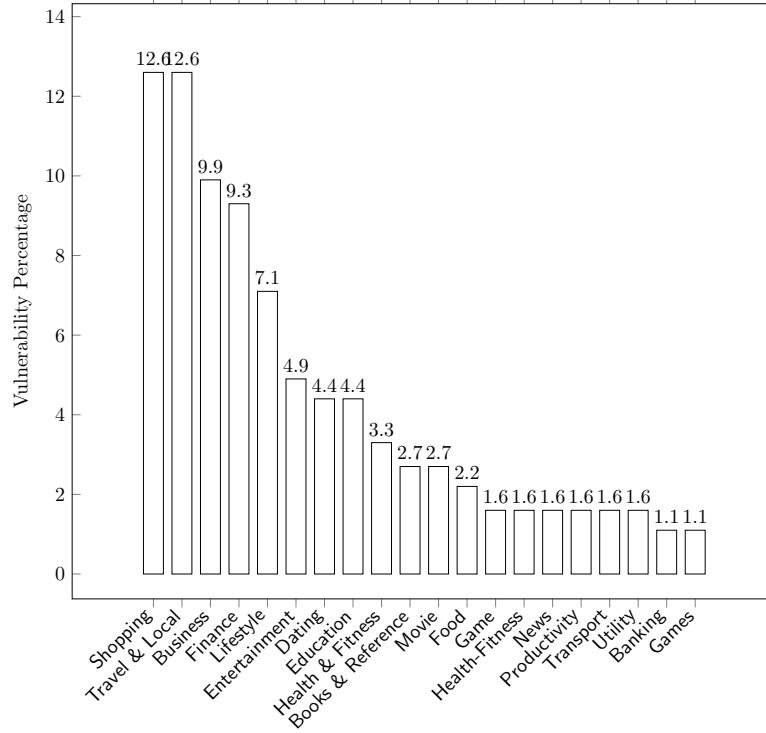


Figure 3.7: Vulnerabilities distribution by categories

& Entertainment, Education, and Finance are the top four sectors which are vulnerable to exploits (*these sectors have been easy prey for the hackers*). The table 3.5 provides the Naming convention for the Application Vulnerabilities.

3.5 Summary

In this chapter, authors have successfully created and demonstrated a practical ‘*Security Analysis and DFR system for Mobile computing devices in an Enterprise environment*’. The proposed system is capable of identifying an app which is downloaded on an Android (phone, VM, or wearable) or an iOS (iPhone or iWatch) from an untrusted source. After identification, the system transfers the tainted app to a pre-deployed server for its security analysis which aims to identify top security threats that could have been exploited by hackers or malicious entities. The solution then offers a process to automate the security analysis, and suggested potential improvements to the existing app forensic analysis techniques. The authors have obtained an prediction accuracy of 94.2 %, after applying machine learning techniques to predict vulnerabilities. The authors found that ‘**M10-Lack of Binary Protection**’ and ‘**M3-Insufficient Transport Layer Protection**’ are the most frequently encountered security threats because the application developers do not perform certificate inspection and fail to incorporate preventive measures against reverse engineering of their apps. Moreover, the apps belonging to ‘**Retail**’ and ‘**Finance**’ sectors were found to be highly vulnerable primarily because these apps have

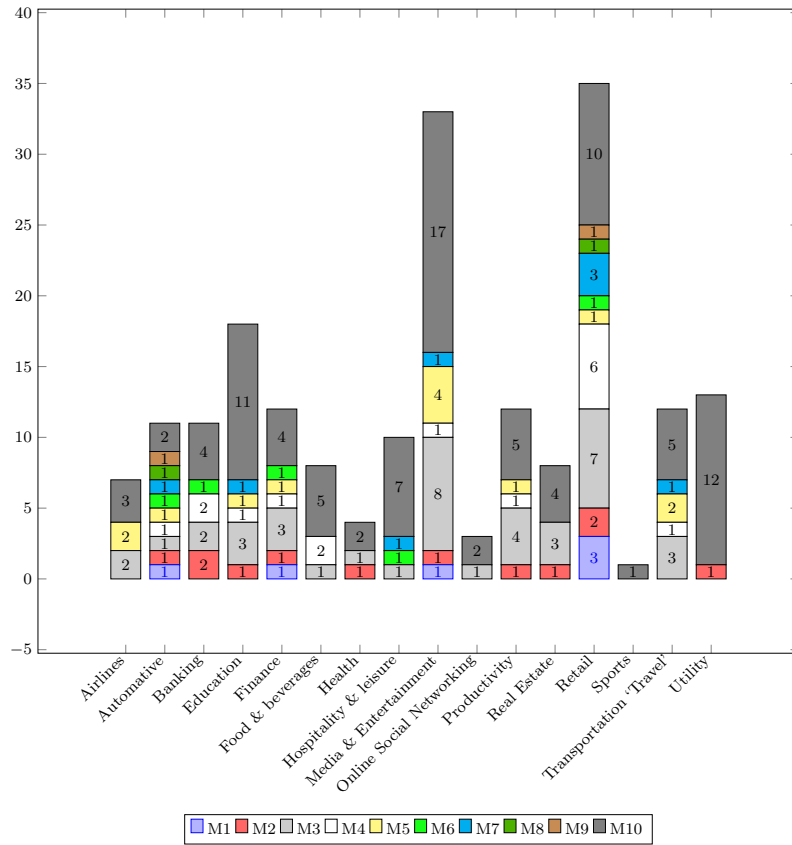


Figure 3.8: Domains versus types of vulnerabilities

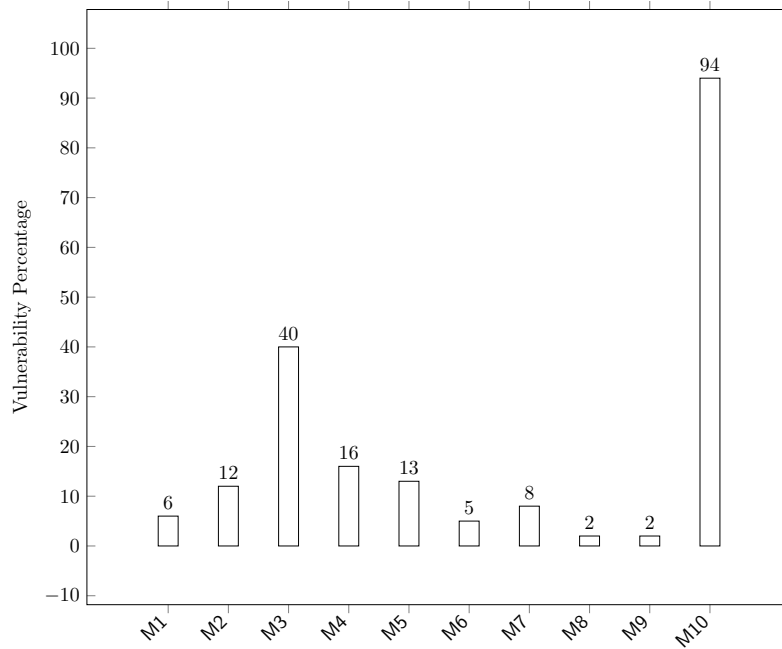


Figure 3.9: Vulnerabilities distribution

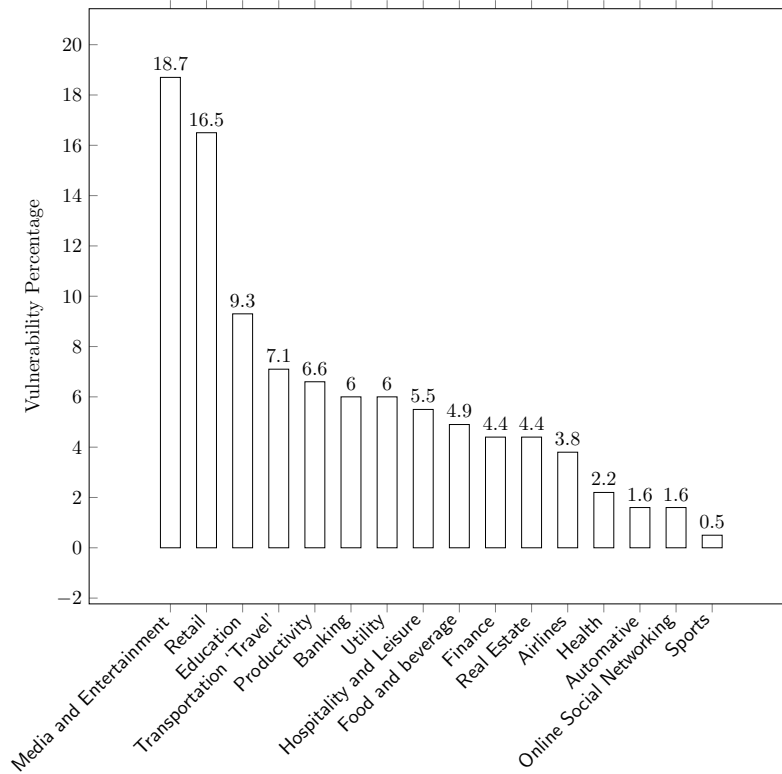


Figure 3.10: Vulnerabilities distribution by domains

high exposure to customer's PII information in addition to finance related information like bank account and card details. The situation gets critical with the increasing volume of customers who have started using these apps from the above-stated domains, in turn attracting more number attackers.

Chapter 4

FORENSIC READINESS FOR IOS OPERATING SYSTEM AND SECURE SOLUTION FOR IOS APPS

In the previous chapter we discussed about the digital forensic readiness system for mobile computing systems, in this chapter we propose iSecureRing, a Forensic readiness and security solution at the application and operating system layer for iOS. iSecureRing solution was presented at IFIP International Conference on Digital Forensics in January 2015 [50] and poster presented at 35th IEEE Symposium on Security and Privacy in May 2014 [53].

Apple's iOS is one of the major players in the smartphone market and it restricts installation of apps that are not from Apple app store. Users often resort to jailbreak their iPhones to break free from these restrictions. Considering jail breaking iPhones is legal in the US; more devices are expected to be jailbroken in future. Jailbroken iPhones are making their way into enterprises, which allow Bring Your Own Device (BYOD), but these devices are either barred or restricted by Mobile Device Management (MDM) softwares, that consider them as a security risk. In this work, authors have designed a solution (iSecureRing) to secure mobile apps and to preserve date and time stamps of events to handle security incidents in the jailbroken iPhones. To the best of author's knowledge, iSecureRing is the first forensic ready mobile app security solution to secure an application running in an unsecure environment within the enterprise environment.

4.1 Introduction

According to a report by Pew Research 2013 [100] [128] 40.98 % of the adult American population owning smartphone use the Apple's iPhone. Apple's iOS does not allow installation of additional applications, extensions and themes that are not available through Apple's App store. Users jailbreak (process to get root level access) their devices to break away from these

restrictions [41]. Once jailbroken, iPhone allows retrieval of applications and corresponding data stored on it, hence posing the risk of compromising the security of the applications and confidentiality of data [115]. As per US copyright office, jailbreaking of iPhone continues to be legal [72]. Considering jailbreaking to be a reality, there is a need to design ways to secure mobile applications running even in the jailbroken iPhones. This requirement of making an application secure in an unsecure environment is critical to the enterprise environment where proprietary application(s) should work without impacting the enterprise security. Currently, enterprises allowing BYOD generally detects and restrict jailbroken iPhones with MDM softwares, like Citrix's XenMobile, IBM's Endpoint manager etc. Employees have to either un-jailbreak their iPhones or use another device to install the enterprise application(s). Solution proposed in this work enables the enterprises to install their application securely on employee's jailbroken iPhone. New or existing apps can be secured and made forensics ready. Forensic readiness of the app enables the enterprises to cross check if it is still running securely, and in case of a security incident there are enough forensics artifacts to investigate.

D'Orazio et al. [35] in their work, authors have proposed "Concealment" technique to enhance the security of non-protected (Class D) data that is at rest on iOS devices, as well as a "Deletion" technique to reinforce data deletion from iOS devices. Hackers, malicious users resort to different techniques like Jail breaking, running the App in debugger mode, Reverse engineering and dynamic hooking in order to gain confidential information stored by iOS application.

Jail breaking – Attackers resort to jail breaking iOS based devices for getting access, once the device is jailbroken, it provides system-level (root) access to the attacker and hence poses risk of compromising the security of the application and confidential data [41].

Debugger mode – Attackers can run the application in debug mode, then get the memory dump and overwrite the memory with malicious code [91] [66].

Reverse engineering– The applications from App Store are encrypted with Apple's Fairplay DRM, which makes reverse engineering of the binaries difficult. Attackers can overwrite the encryption info of the application in jailbroken devices, to get the actual memory dump and analyze the application code for further attacks [115].

Dynamic code hooking – Once the device is jailbroken, it allows attackers to hook the malicious code to the application at runtime to bypass the security checks and functions, thus compromising the security of the application and confidential data [19].

Tampering MAC DTS is an important piece of information for forensics, attackers resort to malicious tampering of MAC DTS by modifying the date and timestamps of artifacts in order

to cover their tracks. In the recent work by Verma et al. [139], the authors have proposed a mechanism for preserving Date and Timestamps for Incident Handling in Android Smartphones.

In this chapter, the author has proposed techniques to detect different hacking attempts to protect the application and data residing on Jailbroken iOS devices. In case of an incident, the author has introduced mechanisms to handle such occurrences for forensic investigation.

The rest of the chapter is organized as follows: section 4.2 provides proposed solution, section 4.3 describes implementation methodology of the solution, section 4.4 describe the Preventing attacks and / or Anti-Forensics, section 4.6 describes the case study of real world implementation and section 4.7 summarize this chapter.

4.2 Proposed solution

iSecureRing to provide static library and the runtime environment for apps to run in unsafe Jailbroken iPhone in an enterprise BYOD environment. The solution consists of two modules:

Securing the Apps - The first module comprises of a static library with secure APIs that can wrap apps in an additional layer of protection, thus making them difficult to crack on jailbroken devices and preventing access to apps data.

Forensic readiness solution (Preserving Timestamps) - Second module to collect and store events with original timestamps in a secure location, the solution preserves original date and timestamps of the events related to the secured app, so that in case of any security incident digital forensic analysis can be performed. The captured timestamps are stored outside the device on a secure server or the cloud. The modules are discussed in detail in following subsections.

4.3 Implementation methodology of our solution

4.3.1 Securing the apps (using static library)

This static library consists of various APIs that can be used to identify security vulnerabilities in jailbroken iPhones. This library can be used to detect and mitigate security issues. The library mainly contains functions namely, `isCheck1()` – iPhone is jailbroken or not; `isCheck2()` – application is running in debug mode or not; `enableDB()` – for disabling the gdb (debugger) for a particular application (process); `isAppC()` – If application’s binary is still encrypted and also check for application bundle files (Info.Plist) integrity; `initialize()` – if any of the static library function themselves are hooked or not; `CheckA()` – critical methods (functions) passed as an argument is hooked or not; `CheckS()` – any methods/functions related to SSL Certificate Validation are hooked or not; `createCheck()`, `createCheckTest()` – to find if the application

is tampered or not; `resetZeroAll()` – to wipe any sensitive data from memory.

4.3.2 Preserving the date and time stamps (using dynamic library)

We have created a Dynamic library using MobileSubstrate framework, this framework provides APIs to add runtime patches or hook to the system functions on jailbroken iOS [121] [122]. The solution architecture (refer Figure 4.1) consists of four components:

- **Dynamic library** – hooks on to the system open calls and captures kernel level date and timestamps corresponding to selected file(s) and then writes them to the log file.
- **Timestamp log file** – the log file is stored in the internal memory of the iPhone which is not directly accessible to applications making it safe against deletion attempts.
- **Uploading the log file** – the log file generated by the DLL, is later uploaded at regular user defined intervals to an external server or cloud based on the network connectivity.
- **The Dynamic library (dylib)** will be loaded into all the running applications. Filters are applied to this dylib so that it gets loaded only for specified applications.

4.3.3 Static library

To protect applications from security incidents, the static library can be used. This library wraps apps in an additional layer of protection that makes apps more difficult to crack in jailbroken devices. This static library contains various APIs that can be used to identify security vulnerabilities in jailbroken iOS devices. The library consists of following functionalities (refer to Table 4.1): Detection of jailbroken devices, Disabling of debugger for an application, Application encryption check (for App store binaries), Detection of dynamic code hooking. The function names used in library are not self- descriptive in order to maintain the code obfuscation that will make attackers difficult to understand the code even when they succeed in reverse engineering the binary.

4.3.4 Dynamic library

In our solution approach we have created a Dynamic library using mobiles substrate framework, now called as Cydia Substrate, developed by a known hacker Saurik [121] [122]. This framework provides platform and APIs to add runtime patches or hook to the system functions and also other applications on jailbroken iOS and rooted Android. Mobilesubstrate framework consists of 3 components, Mobilehooker, Mobileloader and Safemode.

Table 4.1: Static library APIs.

API	Description
isCheck1()	Check if device is jailbroken
isAppC()	Check whether application encryption provided by AppStore is intact
enableDB()	Disables the debugger for an application
isCheck2()	Check if app is running in debug mode
Initialize()	Check if any of these above library APIs are hooked through method swizzling technique
checkA()	Check if the given function is hooked through method swizzling technique
checkS()	Check if any of the SSL validation methods provided by iOS SDK are hooked
makeZero()	Find the data portion of the object's memory and zero it out
encPwD()	Encrypt an object's data in memory with a given secret
decPwD()	Decrypt an object's data in memory with a given secret
listed()	Add the object to a pointer list that wipeAll and checksum operations will be using
unlisted()	Remove the object from the pointer list
resetAllZero()	Wipe all tracked objects
createCheck()	Provide and statically stores a string of all tracked memory addresses and object checksums
createCheckTest()	Test whether the current memory state of all tracked objects matches the state it was in when checksumMem was called

Mobilehooker

Replaces the original function with hooked function. There are 2 APIs that can be used for iOS.

MSHookMessage() – This method is mainly used for replacing objective-c methods at runtime.

MSHookFunction() – This method is used for replacing system functions, i.e., mainly native code written in C, C++ or assembly.

Mobile loader

Cydia Substrate code is compiled as dynamic library and is placed in the path / **Library/MobileSubstrate/DynamicLibraries/** on jailbroken iOS devices. Main task of Mobile Loader is to load these dynamic libraries into the running applications. Initially Mobile Loader will load itself and then it will **dlopen** all the dynamic libraries in above path and loads them at runtime. These Dynamic libraries are configured using Property list files, which acts as filters, controlling whether the library should be loaded or not. This PList file should be of same name as that of dylib and stored along with the dylib on same path. This PList should contain a set of arrays in a dictionary with a key “Filter”. The other keys used are Bundles (array) - The Bundle-ID of the running applications are matched with the list and if any match occurs, then dylib is loaded; Classes (array) - The dylib is loaded if any one of the specified objective-C classes in the list is implemented in the running application; Executables (array) - The dylib is loaded if executable names in the list matches with the executable name of the running application. An example is

given below:

```
Filter = Executables = ("mediaserverd"); Bundles = ("com.apple.MobileSlideShow"); Mode
= "Any";;
```

In above example, the filter is applied such that the dylib is loaded only for iOS built-in application Photos whose bundle ID is 'com.apple.MobileSlideShow' or executable name matches with 'mediaserverd'. When there are more than one filter, Mode key is used. By specifying Mode = Any, dylib will be loaded if any one of the above filters match.

Safe mode

In this mode, all the 3rd party tweaks, extensions will be disabled hence preventing device from entering the crash mode. Then the broken dylib can be uninstalled from the device.

Architecture for the dynamic library

Refer to figure 4.1 for the solution architecture

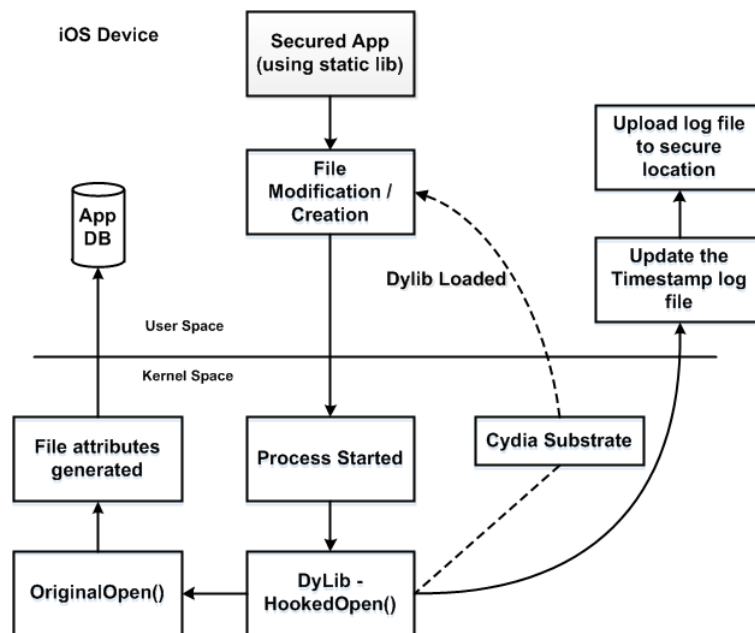


Figure 4.1: Solution architecture preserving date and timestamps.

Procedure for compiling

For editing, compiling and installing dynamic library on the device, we are using a Theos [67], a free development suite. It provides a component called Logos which is a built in pre-processor based library designed to make Mobilesubstrate extension development easy. For compiling, we need to install the Theos on Mac machine. Mac OS X already has most of the necessary

tools required by Theos by default. We still need to install command line tools of XCode if not installed already. Along with Theos, we need to install ldid tool by Saurik. Ldid tool [151] is used to sign the apps or tweaks so that they can be installed on jailbroken iOS devices. To start with the project, we need to get all the iOS private headers of the functions we are hooking. Headers can be dumped using command line tool called Class-Dump-Z [68]. It's a reverse engineering tool which gives complete header information of objective C code in the iOS application. Dumping the headers may take more time as the process includes the headers from all the frameworks including private frameworks also. Once dumped, those headers have to be saved in the folder with respective framework name. Instead of dumping, we can use the headers dumped by others, *ex: rpetrich's headers from github*. All the headers are to be saved in location `/opt/theos/include`. Once this process is completed, we can create the Theos project. In command line, execute the file `/opt/theos/bin/nic.pl` and choose project template, name, etc. We are choosing the project type as library as we want to hook the system function. Once the project is created, a new file called `tweak.xm` will be available within the project directory. This is the place we write our hooking code. Below is the pseudocode for hooking `open()` syscall that is added in `tweak.xm` file:

```
extern "C"
{ int orig\_open(const char *path, int oflags);}
int hijacked\_open(const char *path, int oflags)
{    // do something, then
return orig\_open(path, oflags);
}
%ctor{ NSAutoreleasePool *pool=[[NSAutoreleasePool alloc] init];
MSHookFunction(open(), \&hijacked\_open, \& orig\_open);
[pool drain];}
```

We are using `MSHookFunction()` API to hook the `open()` system syscall. The replacement function is called `hijacked_open()`. Then modify the makefile to add the required frameworks. For this purpose, we are using the Foundation framework for hooking the code. We can add the target, the SDK version and architecture needed to support.

```
TARGET := iPhone:7.0
ARCHS := armv7 arm64
ProjectName\_FRAMEWORKS = Foundation
```

Once done, call make from command line as below.

```
xyz:test xyz make
Making all for application test...
Copying resource directories into the application wrapper...

Signing test...
```

The project is compiled and a dll gets created in `obj` folder.

Loading the DLL on Phone

Once the dynamic library (DLL) is created, it can be installed on the device by Theos suite itself by using the command "make package install." This command will create a Debian package of dll and install in the proper location on the device. Prior, we need to set the environment variable as *export THEOS_DEVICE_IP=iPhone Device IP*. Once Theos prepares the package, it will SFTP (SSH File Transfer Protocol) the package to the device for installation. The device should be on the same network as the computer on which we are developing [19].

4.4 Preventing attacks and / or anti-forensics

4.4.1 Using static library

- `BOOL isCheck1()` - This API is used to check if device OS is jaibroken, and returns YES if iOS in device is jailbroken else NO. This can be called before the application launch.
- API to check debug mode - Application will exit when launched in debug mode with `enableDB()` function. Call this function in `main()` and in other places of the project to disable debugging at any stage. By calling the function `enableDB()` in `main()` or before app launch, the application can be prevented from running in debug mode, this function should be called in Release mode.
- `isCheck2()` - This function gives information about how the application is actually running. If application is started in debugger mode, then 1 is returned else 0 is returned from the function.
- `BOOL isAppC(char* inBundlePath)` - This function checks if application is hacked. The parameter `inBundlePath` can be any character pointer added just for obfuscation, not used inside the function. It includes app encryption check (if the appstore encryption is broken), signer identity checks, etc. If app is cracked, it returns YES else NO. It is mainly to check if appstore binaries are cracked.
- `int Initialize()` - This function checks APIs of static library themselves are hooked or not. This has to be called initially, preferably during app launch to check if these library APIs themselves are hooked by method swizzling technique and take appropriate action. If these functions are hooked then it does not make any sense using these APIs for protecting application.
- `int checkA(const char* MCl, const char* MFr, const char* MFn, void *funcPTR)` - This function checks if any hooking is done for critical method within application that is passed as an argument to this function. Returns 1 if no hook, 0 if any function is hooked. It requires method name, class of the method and path of the framework if framework method or app bundle path if it is application method.

- `int checkS()` – This function checks if any hooking is done for SSL certificate validation methods provided by iOS SDK. This function has to be called within application before calling SSL Validation methods so that proper action can be taken. It returns 1 if not hooked 0 if any function is hooked.
- `makeZero(obj)` - This function is used for zeroing out the variable that is sensitive after its usage
- `encPwd()` and `decPwd()` – These APIs are used for encrypting the sensitive data immediately after they are created and decrypting them only during their usage. Once their usage is done, they can be cleared from memory permanently.
- Using the `listed()` and `unlisted()` functions, several objects can be tracked in order to clear them from memory simultaneously. Different sensitive objects are added to list to keep track of them and later they are cleared at a time with one API. For instance, when the device is locked and/or the app is closed (hidden or terminated) we may want to wipe all sensitive data. We might then add `resetZeroAll()` to the state-change notify functions in your AppDelegate. There are many tools available for attacker to modify the value of some critical data in order to change the behavior of the application at runtime. To track such modifications, we can use APIs `createCheck()` and `createCheckTest()` to create a checksum for critical data and check it periodically to ensure the value of critical data is not tampered by attacker.

4.4.2 Using dynamic library

Whenever files are modified, added, changed the hijacked open call is invoked, in this function the MAC DTS is captured and stored on log file. This log file is stored outside the iPhone on secure location like a server or on the cloud. This logs could be used in offline forensic investigation in case of security incident on the smartphone.

4.5 Experiments and results

We created two apps one without any protection and another using iSecureRing and deployed them on a jailbroken iPhone 4 (iOS 7.0.6). We simulated a series of attacks on the Apps and the data to validate our solution. At the application level, the Apps were subjected to various attacks to exploit lack of binary protection vulnerability [35] as illustrated in the Table 4.2.

The results demonstrate that the App with iSecureRing on a jailbroken iPhone (Row 3, Table 4.2) is as secure as a normal App running in a non-jailbroken iPhone (Row 1, Table 4.2). We conducted performance benchmarking for the 3 cases considered in our experiment, Figure 4.3 summarizes the results from our initial tests (5 runs). The results show that no significant

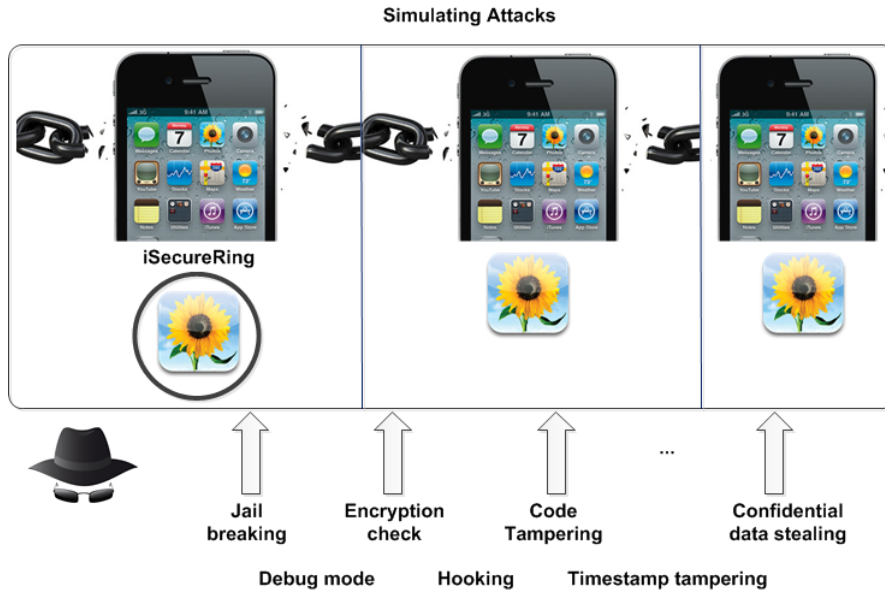


Figure 4.2: Simulating attacks on devices.

Table 4.2: Attacks and results

iPhone 4 (iOS 7.0.6)	Jail breaking	Debug Mode	Encryption check	Hooking	Code Tampering
Non Jailbroken (<i>App with no protection</i>)	Yes	No	No	No	No
Jailbroken (<i>App with no protection</i>)	NA	Yes	Yes	Yes	Yes
Jailbroken (<i>App with iSecureRing</i>)	NA	No	No	No	No

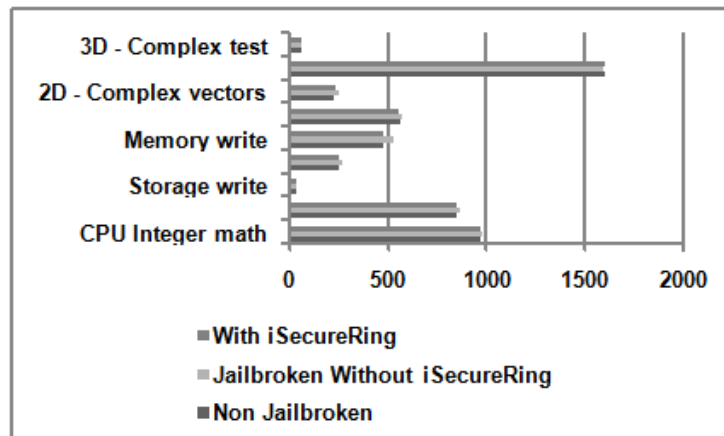


Figure 4.3: Performance benchmark results

```

file=/System/Library/PrivateFrameworks/TextInput.framework/Info.plist
Mtime=Wed Feb 19 11:10:02 2014
Atime=Wed Feb 19 11:10:02 2014
Ctime=Wed Feb 19 11:10:02 2014
Flag=0

file=/var/mobile/Media/PhotoData/Thumbnails/V2/DCIM/100APPLE/IMG_0002.PNG/5003.JPG
Mtime=Sun Mar  2 23:27:24 2014
Atime=Sun Mar  2 23:27:24 2014
Ctime=Sun Mar  2 23:27:24 2014
Flag=0

```

Figure 4.4: MAC DTS logs for an image file

```

May  5 17:12:52 TADMs-iPhone AntiDebugSample[510] <Notice>: MS:Notice: Loading: /Library/MobileSubstrate/Dy
May  5 17:12:52 TADMs-iPhone AntiDebugSample[510] <Warning>: App launched
May  5 17:12:53 TADMs-iPhone com.apple.launchd[1] (UIKitApplication:com.infosysTADM.AntiSam[0xd5c3][510]) <
(UIKITApplication:com.infosysTADM.AntiSam[0xd5c3]) Exited with code: 45
May  5 17:12:53 TADMs-iPhone com.apple.debugserver:300.2[500] <Warning>: 1 -0.000000 see [01fc/1207]: error
18446744069414585344 ) => -1 err = Bad file descriptor (0x00000009)
May  5 17:12:53 TADMs-iPhone com.apple.debugserver:300.2[500] <Warning>: Exiting.

```

Figure 4.5: Preventing debug mode attack

difference in the performance of the device. The iSecureRing also helps in detecting any attempts made to exploit known or unknown vulnerabilities by capturing the timestamps of activities associated with the secured app. We simulated a timestamp tampering attempt on one of the images from Apple's Photo app. iSecureRing successfully captures all the events in the log. Using the log we were able to identify the tampering attempts. Figure 4.4 illustrates the MAC DTS (Modified Accessed Created Date and Time stamps) [7] captured for one of the images.

Debug mode attack – Figure 4.5 is the screenshot of xcode running application in debug mode [10] with our solution iSecureRing the application is exiting protecting the applicaiton.

Encryption check – We used Clutch [12] to simulate this attack. Clutch can be downloaded from Cydia Source cydia.iphonecake.com. iSecureRing includes a check to determine if application encryption is intact by analysing the encryption information in the binary and also the cryptid flag value. If application encryption is found to be broken, then the user can be alerted and application behaviour can be changed at runtime.

Hooking – We simulated hooking attack by hooking SSL validation methods, one can do MITM attack on even HTTPS requests to understand, steal and tamper the request data. iSecureRing includes different checks for identifying hooking of critical methods like SSL Validation, authentication, etc. Applications using iSecureRing can be protected against such attacks as it alerts the user and changes the app behaviour in such attack scenarios.

Code Tampering – Simulated code tampering attack by using a tool Cycrypt [123], it is a javascript interpreter. This tool is used to modify the iOS application behaviour at runtime like bypassing some authentication checks, accessing critical information from memory, etc. Application compiled with iSecureRing, provided different APIs for identifying code tampering

scenarios for critical instance variables or class objects with CRC checksum comparison and also provides APIs to wipe the sensitive data from memory.

4.6 Case study

iSecureRing solution was successfully implemented for one of the leading banks for securing their iPhone Mobile application.

Problem statement – The bank had a security incident in which if the device is jailbroken, the application was revealing sensitive data.

Jailbreak attack – The attacker had used latest jailbreaking tool for iOS 7.1.2 Pangu [17]. Further attacker had used tweaks to bypass any jailbreak detection. There are some tweaks/apps available in Cydia which will bypass the jailbreak detection check within the application, and make application run normally even on jailbroken devices. One such tweak is xCon [71], a dynamic library developed by n00neimp0rtant and Lunatik, which hooks the low level APIs used for jailbreak detection like file related APIs and other system calls, thus bypassing the jailbreak detection functions used within application. There is no configuration required for xCon. It's a mobilesubstrate dynamic library that can be installed from Cydia.

iSecuring solution results – We secured the mobile app using our iSecureRing solution. Our jailbreak detection function was robust to detect if the device was jailbroken and further if any jailbreak by pass functions as well. We tested our solution with xCon 39 beta 7 on iphone 4s with iOS version 7.1.2. xCon was not able to bypass the jailbreak detection techniques used in our solution. Our solution checks for the hooking of any of file related system function existence thus making xCon like tweaks useless. Our solution mitigates the jailbreak detection bypassing mechanism provided by xCon dynamic library

4.7 Summary

In this chapter, authors have demonstrated that their solution helps in securing the apps on a jailbroken devices. By utilizing the static library, we could detect security vulnerabilities and take appropriate actions. We have also shown how authentic copy of the MAC DTS values can be preserved on a local server or the cloud, which can help in offline digital forensics investigations in case of a security incident. Thus with our proposed approach, any new or existing apps can be secured and made forensics ready, even if the iOS device is jailbroken. This is the main contribution of this work. With jail breaking on the rise and expected to continue, our work helps in addressing the security gap that exists. With enterprises adapting BYOD and jailbroken devices making its way into enterprises, our solution would help enterprises mitigate the security risks. This enables employees to use one device both for official and personal usage. Further with our Dynamic library we have proposed a solution to detect malicious tampering of data by storing an authentic copy of the MAC DTS values of the artifacts on a secure location outside

the iOS devices.

Chapter 5

FORENSIC READINESS FOR ANDROID OPERATING SYSTEM BY PRESERVING DATE AND TIMESTAMPS

In the previous chapter we discussed about the Forensic readiness and security solution for iOS at the application and operating system layer. In this chapter author discuss Forensic readiness solution for Android operating system. This work was presented at IFIP International Conference on Digital Forensics in January 2014 [139].

Enterprises save money and time when they allow the employees to carry their own computing devices to the workplaces. The employees find it convenient to work on a single device for professional as well as personal use as it is easier to manage data in one place and it also improves their efficiency. This phenomenon of using personal devices at workplaces is called Bring Your Own Device (BYOD), and it is rapidly being adopted by enterprises around the world; however, securing personal and professional data on the device is still considered to be a huge challenge for both employees and employers. People with malicious intent can try to get access to this valuable data and/ or tamper with it. In this chapter, a proactive approach has been proposed to address this challenge, which gathers kernel generated timestamps of events and stores them in a secure location outside an Android smartphone. In the case of a security incident, these stored timestamps can help in the offline digital forensic investigation. To the best of our knowledge, this research work is the first attempt which preserves authentic timestamps of events on an Android operating system to detect potential malicious actions, including anti-forensics.

5.1 Introduction

Smartphones growth and adoption have gathered pace during the past five years. According to a Gartner's report [60], during the second quarter of 2013, the sales of smartphones have surpassed the sale of feature phones and accounts for 51.8% of the market share, which is around 225 million units. In the same report, Gartner says that Android¹, an open source mobile operating system, leads the smartphone operating system (OS) market with 79.0% share. International Data Corporation (IDC) says that Android would dominate smartphone operating system market during the year 2013 with around 75.3% of the share and would continue to dominate till year 2017 [97]. Both of these reports confirm that Android is the leading player in the smartphone market today and is likely to stay there in the near future.

In the survey report (May, 2013) by the Pew Research Centre says that 61% of American cell phone owners have a smartphone, out of which 28% own an Android powered device [129]. Most people prefer to have one device for both personal and professional needs, so instead of taking a new device from their employers, people take their personal device(s) to their workplaces. This has given rise to a new trend i.e. Bring Your Own Device (BYOD) [93] which is catching up very fast across the world. According to another report by Gartner, 50% of the companies will allow their employees to carry their own devices by 2017 [144], and majority of it would be smartphones and tablets. In the enterprise environment, BYOD helps employers to save on the cost of devices, whereas the employees find it convenient to manage their personal and professional data on a single device. However, BYOD also has led to a new set of mobile security threats opening up new windows of opportunities for fraudsters to access, modify, edit, steal information and cover their tracks by tampering data on the device. We think this is one of the weakest link for getting unauthorized access to enterprise networks and carry out malicious actions.

A mobile device is personal to a user, so is the data residing in it. These data are invaluable and personal to a user and hence attempts to get unauthorized access to it are not uncommon. In BYOD environment, the incentives of stealing information from the phone are high as both personal, as well as confidential corporate data, is stored on the same device [93]. Access control restriction on files can protect against unauthorized access from someone other than the owner of the device; however, if the owner herself wants to compromise the security of data, she can do it. People with malicious intentions could try to hide their actions by tampering the metadata information including Modification, Access, Change and/ or Creation Date and Time Stamps (MAC DTS) of accessed data to match MAC DTS that existed before malicious access. We have proposed a solution that can detect such tampering with timestamps.

Timestamps reveal the information regarding when a file was created, modified and accessed,

¹"Android, the world's most popular mobile platform," accessed September 14, 2013, <http://developer.android.com/about/index.html>

which plays a significant role in the reconstruction of the sequence of events in digital forensics. Criminals tamper digital data to either destroy the evidences or try to plant evidence in order to implicate others (called anti-forensics, *section 5.3.1 and 5.4.7*), but authentic timestamps of these actions would help investigators in reconstructing the sequence of events.

Weil et al. [143] proposed that dynamic time and date stamp analysis can determine the actual time of events on a PC (especially when they are not available or tampered one or more times) by utilizing external source of time and date. Carrier et al. [21] suggested that a dynamic timeline analysis is essential in the reconstruction of digital events, which should be implemented after a security incident has taken place in order to find the person responsible. To implement the same ideas a customized forensic tool ‘Zeitline’ was created by Buchholz and Falk [20], which could incorporate timelines from various sources and presented details through a graphical interface. They had a limitation in handling clock differences and clock drifts. Another more improved tool was created in 2009, by Olsson et al. [101] named Cyber-Forensic Time Lab (CFTL) which was more intuitive to use with its graphical interface. A different approach to address the same problem was given by Marrington et al. [87] where they traced Computer Activity Timeline by utilizing the “causality” of events on a computer system. The tool extracted the MAC timestamps from the Hard Disk Drive (HDD) and then correlated the events according to their causality. The problem of MAC timestamps tampering has not been addressed in their work which can cause irregularities in their results.

MAC DTS of digital data on a smartphone could be very crucial and the most fundamental evidence in digital forensic investigations, thus establishing authenticity of available MAC DTS is of prime importance for forensic investigators. Malicious tampering of MAC DTS mainly comprises of modifying date and time stamps and/ or contents of the files. Commercial tools including Cellebrite Universal Forensic Extraction Device (UFED) System, Forensic Tool Kit (FTK) Mobile Phone Examiner and other mobile forensic tools provide solutions to recover data; however, most of them prove to be inadequate when trying to establish the authenticity of MAC DTS. The novel method for logging MAC DTS of file system started with the work of Barik et al. [14]. Das et al. [29] and Ansari et al. [9] have worked further on this strategy to use it in File-system intrusion detection system. These three papers show work on Linux operating system with storage media is a conventional hard disk drive. We carried forward this approach to the Android operating system, which looks like Linux, but has lots of variations in implementation and working. The storage media and the way file system stores metadata information is also quite different.

Recent work by Grover[57] proposed an enterprise monitoring application, ‘DroidWatch’, which continuously collects data sets from an Android phone. The collected data sets are uploaded to an enterprise server which can help security personnel for monitoring, auditing, responding to an incident and carrying out a forensic investigation. Their solution is not secure against root

attacks, and application uninstallation attacks.

Another related area is Android phone anti-forensics. Distefano et al. [33] have shown an anti-forensics technique in their work where they use the private folder of an installed Android application to hide counterfeit potential evidence. They propose a safe uninstallation process to cover up the tampering. Another improved approach was implemented in the work of Albano et al [5], where they restored the timestamps of tampered files without raising any suspicion and without leaving any change in the file system. But this method could only change the contents of the storage media and therefore the data uploaded outside of the device remains untouched.

Linux Security Modules (LSM) were created as a framework to support security modules [126]. Security modules that have been implemented include Security Enhanced Linux (SELinux), AppArmor, Smack and TOMOYO Linux. In their work, Shabtai et al. [125] implemented SELinux on Android platform to perform strict access control and strengthening the security of Android. The steps included compiling the kernel with SELinux support, followed by designing an Android particular security policy. The standard startup processes and scripts to support were modified for loading the new policies at startup, and finally the Android disk image (popularly known as ROM) was created ready to be flushed on the phone.

The National Security Agency (NSA) built upon their work by creating a standard for implementing SELinux on Android devices, which they named as "SEAndroid" [127] in early 2012. SELinux enabled Android was tested against most contemporary exploits which aim at gaining root access. Both [125] and [127] required the disk image on Android phone or tablet to be flushed with their own customized ROM, which is impractical in a BYOD environment. Whereas, in our solution we have focused on preserving the MAC DTS without any reflashing which keeps the phone data intact, more suited to the BYOD environment. In the process, we also ensure that the device is kept intact with user's personal data so that regular usage can be continued without any impact after the LKM's installation.

In this chapter, authors first demonstrate that tampering of MAC DTS on Android smartphones is possible (an anti-forensic approach) by taking two real world scenarios. In the next step we design a mechanism for preserving date and time stamps for incident handling in Android smartphones. The system generated MAC DTS values are captured with the aid of a Loadable Kernel Module (LKM) [64] which hooks into the system calls to catch these values. Thus the solution aims to detect such malicious actions by capturing the date and time stamps along with location details and store a log of events in a secure place outside the phone. A secure place outside the phone can be a cloud or a local server in the enterprise environment. We, therefore, propose a proactive approach to analyze the MAC DTS Values, where a snapshot of authentic MAC DTS stored at the secure place is used when a security incident has to be investigated for validating

the authenticity of the files in question.

The rest of the chapter is organized as follows: section 5.3 talks about tampering of MAC DTS on Android device. Detection methodology and implementation is described in section 5.4; performance implication is analyzed in section 5.5 and section 5.6 provides the chapter summary.

5.2 Android v/s iOS preserving timestamps comparison

In the previous chapter the author discussed about the preserving timestamps for iOS, in this chapter the author discuss similar approach for Android. The table 5.1 provides the comparison of the techniques for iOS versus Android Operating systems.

Table 5.1: Preserving MACDTS iOS versus Android Operating systems

Technique	Android	iOS
Approach	Created a Loadable Kernel Module (LKM) which hooks onto the sys_open() system call and captures kernel level timestamps values corresponding to selected file(s), and then uploads the timestamps to a secure local server	Dynamic library using MobileSubstrate framework, this framework provides APIs to add runtime patches or hook to the system functions on jailbroken iOS
Trapping the system calls	LKM algorithm mainly consists of four functions namely root_start, root_stop, hacked_open and hacked_unlink. root_start initializes the log file pointer, points the sys_open() call address to our hack_open() call	Mobilesubstrate framework consists of 3 components, Mobilehooker, Mobileloader and Safemode. Mobilehooker replaces the original function with hooked function MSHookFunction() API to hook the open() syscall. The replacement function is hijacked_open().
Capturing the events	hack_open function takes the file path name and file attributes as the input parameters, and if the file path matches with the selected file or folder in the filter array then the date and time stamp details are written to a log file, after this operation is completed the control is passed back to the original sys_open() call.	Whenever files are modified, added, changed the hijacked open call is invoked, in this function the MAC DTS is captured and stored on log file

5.3 Tampering of MAC DTS on android phone

We started with development and testing of an attack tool (section 5.3.1) which can tamper files and their timestamps so that no evidence is left behind for a reactive forensic investigation. We developed our solution (section 5.4) which can detect such activities. We have worked with HTC Wildfire and Samsung Galaxy S2 GT-I9100 Android phones. The two phones were chosen as they represent two ends of smartphone segments in terms of processing power, storage capacity, battery power and cost. HTC Wildfire falls under the category of low end smartphone with 0.60

GHz CPU speed, single core processor, 512 MB RAM, and 1230 mAh battery capacity. The Samsung Galaxy S2 falls under high end smartphone category with 1.20 GHz CPU speed dual core processor, 1GB RAM, and 1650 mAh battery capacity.

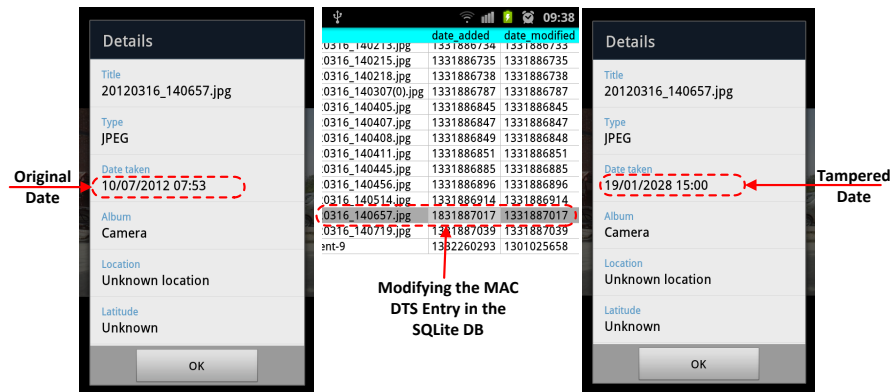


Figure 5.1: Tampering of date and timestamps.

Android file system stores data and their corresponding metadata (including MAC DTS values) into ‘SQLite’ databases. It includes phone calls, short message service (SMS), multimedia messaging service (MMS), photo/ video captured by camera, audio recorded by the microphone and all other types of text files (word, excel, pdf, text etc.). User applications access these data through various API’s and interfaces that are controlled by the OS. Moreover, all applications need to have explicit permissions to use these API’s and interfaces which are generally granted to them at the time of their installation. The android file system ensures no direct access to these SQLite databases is provided to the user, by storing them into the restricted internal memory of the phone.

Although, Android is claimed to be very secure by design, literature is available which shows there are ways to bypass Android security [80] [16]. In the Android market, we found several time stamp tampering applications, for instance applications that can modify the timestamps of received SMSes by overwriting ². We also built our own application that can tamper time stamps on an Android phone for calls, SMSes, camera images and video files. Figure 5.1 on previous page, shows the screen shots of a camera image file before and after tampering timestamps.

5.3.1 Attack methodology and/ or anti-forensics

All actions that can meddle with data present in a particular digital storage device, with an aim to destroy or tamper the potential pieces of evidence present on the storage are called anti-forensic activities or actions. We are using the anti-forensics classification used in [62] and later in [33], namely destroying the evidence, hiding the evidence, altering the evidence and counterfeiting the evidence. Our attack methodology involved creating a MAC DTS tampering application which do the following:

²“SQLite Editor,” Google Play, accessed September 14, 2013, <https://play.google.com/store/search?q=SQLite%20Editor>.

- Eliminating / altering the evidence: creation of evidence is prevented or a slight alteration is made so that data collection during a forensic investigation can be thwarted. The MAC DTS tampering application extracts data entries from the SQLite database through APIs into a text file. Data in this file is changed in such a way that the database entry loses evidence value. Example, the ‘type’ column entry of some call-entry in *calls* table (inside *contacts2.db*) is changed to ‘5’, whereas the default values are ‘1’ for incoming call, ‘2’ for outgoing call and ‘3’ for missed call. After this our tool writes the text back to the database. A forensic tool extracting entries from this database would miss the edited entry while extracting call records out of the device. This is just one example, similar operations can be performed on other potential evidence containing SQLite databases.
- Counterfeiting evidence: implanting fake evidence; the application can overwrite entries in SQLite databases. It can also overwrite the logcat buffer,³ which stores the debugging logs, by attacker’s own version.
- Destroying the evidence: deleting the evidence; the application can also delete the contents of SQLite databases and the logcat buffer.

In above three cases, data retrieval is possible but not guaranteed. It is costly as well as time consuming [45] [113]. In the next two sub-subsections, we have demonstrated attacks of similar kind in two real world scenarios.

Scenario # 1

In this scenario, assume a person has left his phone alone for a short duration (say 5-10 mins) and the attacker gets physical access to it. Following are the steps we followed:

- We connected the phone to our laptop which had Android SDK installed (open source development tool for Android) in it.
- Installed our MAC DTS tampering application on his phone. Even if the phone is locked we can still install our application. Like all contemporary mobile forensic tools, our solution also requires ‘USB’ debugging mode enabled.
- Extracted the contents of camera image database into the internal memory folder of the application and then pulled the file into our laptop.
- We modified the metadata and its MAC DTS.
- We pushed it back on the phone and updated the new values back to the database.

³“Logcat,” Android Developers, accessed on September 14, 2013, <http://developer.android.com/tools/help/logcat.html>

- Although the process looks tedious, we only had to run a set of simple commands (which can otherwise be combined into a script). The above tasks took us an average time of 3 minutes (5 iterations) to complete which is enough time to carry out the attack in the real world.
- We cleared the logcat entries. Physical acquisition from a mobile forensic tool can reveal the deleted contents, but would be time consuming and costly [113].

Scenario # 2

In the second scenario, assume an employee working for a company wants to steal a confidential document. The time at which he accesses the document can lead to suspicion. For example, suppose five people working in XYZ Corporation have the quotation of a big business deal on their phones which is opening for bidding the next day. One of them happens to access the quotation at 2 AM in the morning with the intention of sending it to a rival firm. In this case, he would like to hide his tracks of accessing the file on his phone by modifying the MAC DTS values of that document. Following are the attack steps he can follow:

- Installs our MAC DTS tampering application on the phone.
- Accesses the confidential file and steals the information.
- Modifies the MAC DTS values to same as they were before the access, in order to hide tracks.
- Uninstalls the tampering application and clears the logcat entries.

5.4 Implementation methodology

In our solution approach, we have created a Loadable Kernel Module (LKM) which hooks onto the `sys_open()` system call and captures kernel level timestamps values corresponding to selected file(s), and then uploads the timestamps to a secure local server. LKM [26] is supported by all versions of Android.

The detailed algorithm for the LKM module is explained in subsection 5.4.1. Procedure for compiling the LKM is described in subsection 5.4.2, followed by steps for loading the LKM on the Phone in subsection 5.4.3. After the LKM is loaded it starts reading the MAC DTS values from the phone. The captured values are written into a temporary log file which is stored in the internal memory of the phone. As per availability of network connectivity, the log file is uploaded to a local server (subsection 5.4.5). Figure 5.2 above, provides the implementation architecture for our solution.

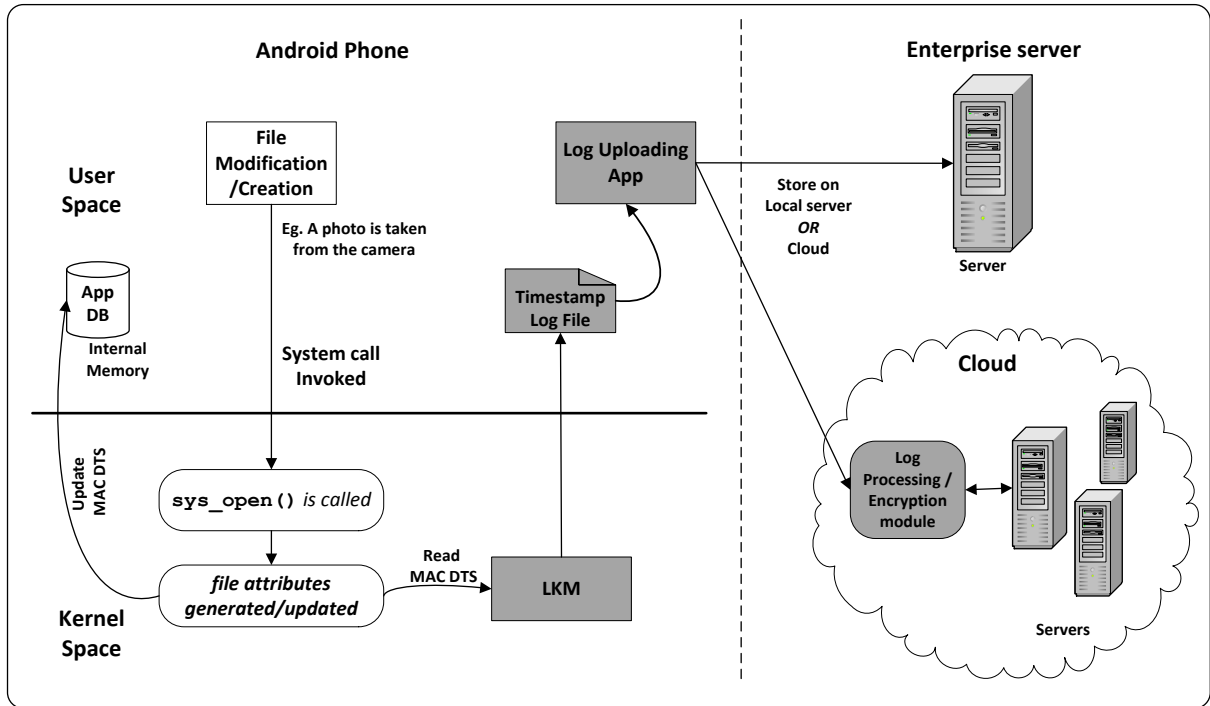


Figure 5.2: Solution architecture –preserving date and timestamps.

5.4.1 Algorithm for the LKM module

LKM algorithm (refer to appendix 1) mainly consists of four functions namely `root_start`, `root_stop`, `hacked_open` and `hacked_unlink`. `root_start` initializes the log file pointer, points the `sys_open()` call address to our `hack_open()` call and points the `sys_unlink()` call address to our `hack_unlink()` call. `hack_open` function takes the file path name and file attributes as the input parameters, and if the file path matches with the selected file or folder in the filter array then the date and time stamp details are written to a log file, after this operation is completed the control is passed back to the original `sys_open()` call. The `hack_unlink` function takes the file path name as the input parameter, and if the file path matches with the log file path then denies delete operation else the control is passed back to the original `sys_unlink()` call. The `root_stop` function closes the log file pointer, restores the address back to the original `sys_unlink()` call routine and restores the address back to the original `sys_open()` call routine.

5.4.2 Procedure for compiling the LKM

The LKM has to be compiled against the kernel source code on which it has to be loaded. Initial step consists of setting up the kernel environment on a computer. We found the instructions for these steps on Android Open Source Project's website ⁴. The next step is to acquire the original kernel source code of the Android version installed on the device. We downloaded Android 2.1 'Eclair' kernel source from HTC's website [27] and similarly Android 2.3 'Gingerbread' kernel

⁴"Downloading and Building," Android Developers, accesses Sep 14, 2013, <http://source.android.com/source/building.html>

source from Samsung’s website [28].

Table 5.2: List of smartphones used in our experiment.

Phone	Android Version	Vermagic string	Address of sys_ call_table
HTC Wildfire	2.1, Eclair	2.6.29-4266b2e1	0xc002bfa4
Samsung Galaxy S2 GT-I9100	2.3 Gingerbread (XWKL1)	2.6.35.7- I9100XWKL1- CL809037	0xc0028fe4

After that, we had to find the physical address of the system call table on each phone to hook on to the system calls. Another mandatory condition for loading the LKM successfully on the phones is that “vermagic string” (refer Table 5.2 for details) of both their kernels and the LKMs should match [108].

5.4.3 Loading the LKM on phone

Root permissions (administrative privileges) are required for loading the LKM. Generally temporary rooting is a good option as it involves the least number of changes to the file system, which is preferred in digital forensics. We decided to apply temporary rooting on Samsung Galaxy S2; however, we permanently rooted the HTC wildfire to explore how our solution performs on permanently rooted devices. Our solution works seamlessly on both temporary and permanently rooted devices. We followed rooting instructions from a mobile software development community portal - *XDA Developers*⁵.

After rooting of the phone, the LKM was loaded. The LKM monitors the events happening on the selected set of files or directories. The module can read all the MAC DTS values for these files and can write them to a log file (recordfile.log, refer figure 5.3 on next page) stored in the internal memory of the phone.

5.4.4 Denying deletion of log file

The log file is stored in the internal memory of the Android smartphone which is not accessible directly to applications, and hence the file is secure against deletion attempts. In case attacker manually finds the log file, any attempt to delete it is denied by the LKM module bypassing call to `sys_unlink` (Appendix 1, `hack_unlink()` call) thus ensuring security of the log file. This mechanism can be understood as a measure of **anti-anti-forensics**, which means that by not allowing the deletion a possible destructive anti-forensic attempt has been foiled.

⁵XDA Developers, accessed September 14, 2013, <http://www.xda-developers.com>

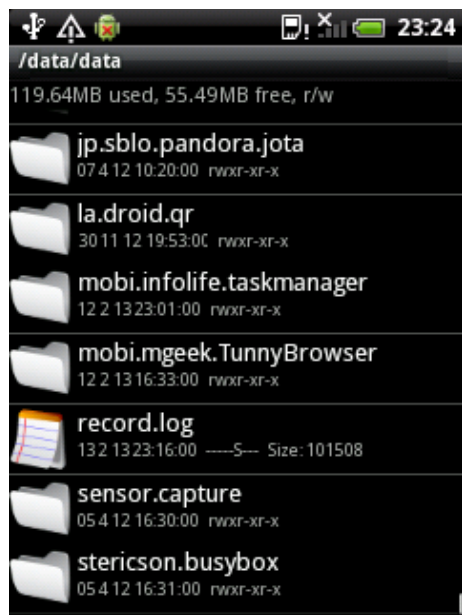


Figure 5.3: Storing *record.log* file in internal memory.

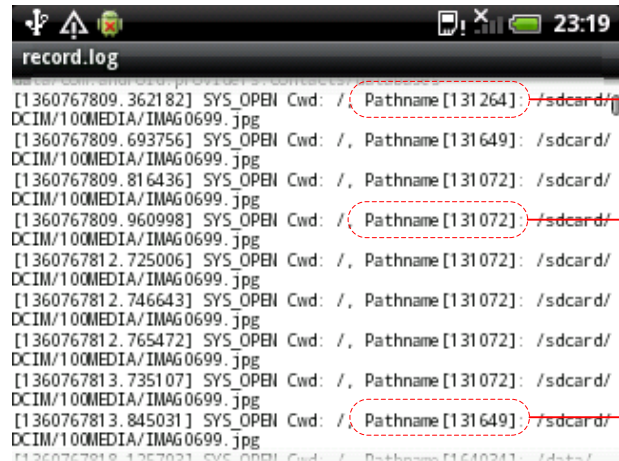


Figure 5.4: Snapshot of the *record.log* file for camera image 'IMAG0699.jpg'.

5.4.5 Uploading the log file

The log file generated by the LKM is later uploaded at regular intervals, depending on the availability of network connectivity, to a local server with the help of our uploader application. We used the WAMP software stack on our local server which runs Apache, PHP and MySQL. The system can easily be extended to a cloud which can act as the external storage where the logs can be uploaded. The MAC DTS generated entries for every file (Figure 5.4, above) can be distinctly identified on the basis of unique flag values associated with operations like file creation, modification or deletion respectively.

The LKM can read MAC DTS values for individual files like image files, audio files, text files and SQLite databases etc. However it cannot capture timestamps for individual database transactions. For every SMS and call a record is added to their respective databases, whereas all images, audio, video and other common files are updated directly on the storage device as well as entries added to their corresponding databases. For this reason along with uploading the log file, the uploader application can also extract and upload specific details (like the called numbers, call status i.e. incoming or outgoing, and the call/SMS time) from 'call-records' folder and 'SMS' databases to track which entry has been updated in these databases. The uploader application also writes the cell tower information and the GPS location of the phone into the log file before uploading it. Adding location information to the log file enhances the credibility of the logged MAC DTS values as it reveals the location at which probable tampering could have taken place.

For addressing privacy issues arising due to storing the log file on the third party cloud server, a middleware component running on the cloud which can encrypt the log file before storing on the cloud server can be created.

5.4.6 Incident handling

A phone on which a security incident has occurred and the tampering of MAC DTS is suspected, provided the solution is already implemented, then the reactive approach can be used to verify their authenticity. The MAC DTS values of suspected file(s) on a phone are cross-checked with corresponding MAC DTS log values stored on local server or cloud. If the values match then no tampering has been done otherwise there is some discrepancy present.

In some cases tampering can also be detected even if the solution had not been deployed on the device, or if there is some discrepancy that was present on the phone before the solution got operational. For instance, all the SQLite databases on a device have a primary key (named `_id`) which increases in sequential order as and when a new record is inserted into it. If a call is made or an SMS is sent to the already existing number, a new record is written to the database. Camera images, videos and audio files etc., also have their separate records in their respective databases, and in case they are edited (by a image, video or audio editing application) a new entry with new `_id` is written into the database. If the timestamp values of one record having lower `_id` value are greater than another record with higher `_id` value (means that it was inserted later), then timestamps values have been tampered for sure. These results can then be forwarded to a digital forensic investigator for further examination.

5.4.7 Anti forensics analysis

Apart from all the techniques specified in section 5.3.1, we are considering an additional category of ‘detecting forensic tools which are working on the device’ which was proposed by [11]. While trying to *destroy evidence*, the first thing an attacker can do is delete the evidence file itself. Our implementation cannot recover deleted data, but it can tell the time of deletion of a file. Secondly, if the attacker tries to tamper with the metadata of the evidence file, in that case, the solution can detect it and give the original values from stored logs. Thirdly, if the attacker somehow figures out the location of our log file he may try to delete it, however, deletion of log file would not be supported by the operating system (refer to section 5.4.4).

Our solution is resistant to *hiding evidence* anti-forensics, because if the LKM is running, any kind of log redirection is not possible. In case the attacker wants to *alter evidence* and *counterfeit evidence* using technique explained in [5], the LKM will get unloaded and logging would stop. This would result in no log file uploads to the local server. There can be a provision made at local server that, if no update is received till a fixed amount of time an alarm can be raised which

could be used to isolate the device for further examination. *Detection of forensic tool running* is not directly applicable to our solution as activities, which have occurred after its installation, only are captured. As the enterprise deploys our solution on a device before it enters their ecosystem, everything that happened earlier is not a matter of concern for the enterprise.

Table 5.3: Performance benchmark parameters

Test Parameter	Operations
RAM read/ write operations	MB per 10 seconds for Integer array copy and sum into the RAM
CPU Integer	Million operations per 10 seconds
CPU Float point	Million operations per 10 seconds
2D graphics	Performance evaluation for doing 2d animation
3D graphics	Performance evaluation for doing 3d animation
Database I/O	Performance of SQLite queries like INSERT, SELECT and UPDATE etc.
SD Card write operations	MB of data written to SD card per 10 seconds
SD Card read operations	MB of data read from SD card per 10 seconds

5.5 Performance implications

To test the impact of installing the LKM on the phone, we ran 30 iterations of the ‘System Test’ benching tool ⁶ on the HTC Wildfire smart phone with and without the LKM. We selected this tool specifically, because it provides results for various system peripherals at the same time. The system components considered in benchmarking are stated in Table 5.3 (above), and the graph given in Figure 5.5 (below) summarizes the results. We recognized that there is no significant difference in the performance of the phone. Similar results were observed in the case of the Samsung Galaxy S2.

⁶System Test, Google Play, accessed September 14, 2013, <http://goo.gl/0f67R>

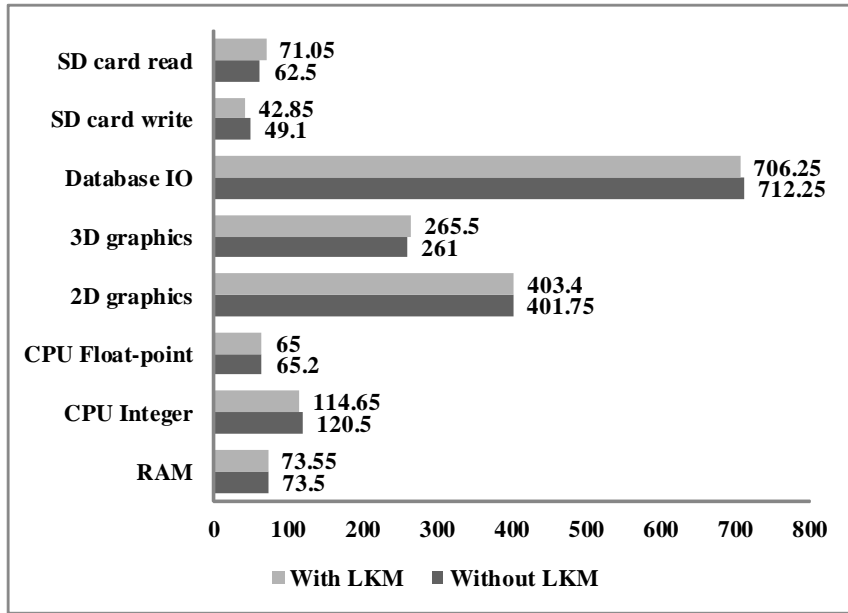


Figure 5.5: Performance benchmark results on HTC Wildfire

5.6 Summary

In this chapter, authors demonstrated that Android smartphones used in a BYOD environment are vulnerable to confidentiality and integrity attacks, where attackers can gain access to data stored on the phone and tamper it. Proposed solution helps to detect this malicious tampering of data by storing an authentic copy of the MAC DTS values for selected files/ directories on a local server or the cloud and for later verification / validation of the MAC DTS of questioned files/ directories. Android based devices are catching up as personal devices and finding their way into offices (BYOD). This solution can be applied on any Android powered mobile computing devices. The concept presented in this chapter, which uses kernel generated authentic timestamps stored at an outside secured place for identifying malicious activities, can be extended to other mobile device operating systems.

Chapter 6

FORENSIC ANALYSIS OF MOBILE DEVICE ADS TO IDENTIFY USERS

In the previous chapters, we discussed the digital forensic readiness system for mobile computing systems, Forensic readiness and security solution at the application layer and operating system layer for iOS and Forensic readiness for the Android operating system. In this chapter the author proposes a novel approach to Forensic analysis, analyzing Mobile Device Ads to Identify Users. Authors presented this work at IFIP International Conference on Digital Forensics in January 2016 [51].

User's browsing behavior is tracked by the search providers to construct his or her activity profile. The tracking helps the search engines to fine tune the results and presenting user specific advertising. When a search input matches with some commercial product or service offering, ads based on the previously saved interests, likes and dislikes are displayed. The number of web searches from mobile devices have exceeded that of desktops. People are using mobile devices for critical business tasks like e-commerce, banking transactions, video conferences, email communication and storing confidential data. Companies are moving towards mobile app-only strategy. The advertisers are now displaying ads on mobile apps as well. Security incidents on mobile devices are on the rise, and they are frequently being used in committing criminal activities. Ads many times can reveal information such as location, gender, age and other valuable data about the user. This chapter aims to extract ads on mobile devices, retrieve user-specific information to reconstruct the user profile and to predict the user's identity. The authors have shown that, with their system and method based on the analysis of ads, a user can be identified even if he or she is using the same device, multiple devices, different networks or following different usage patterns. The proposed system can be used as one of the techniques in digital forensic readiness framework for mobile devices. The system has a wide range of applications in context-based security, proactive and reactive digital forensic investigation.

6.1 Introduction

Mobile devices can be taken anywhere, increasing the possibility of getting stolen or physically tampered. Smartphones enable the users to access the internet from anywhere. Mobile devices are vulnerable to remote attacks through SMS/MMS or by exploitation of the insecure connection. As per checkpoint mobile security survey report 2014 [31], 82% of security professionals expect mobile security incidents to increase this year, 98% have concerns about the impact of a mobile security incident and 95% face challenges with the security of BYOD (Bring Your Own Device).

Unlike hard disk drives, it is not possible to forensically image a phone without changing the state of the original. Since they use the flash memory, everytime extraction is made, the hash is different. All forensic “images” of phones are not created equal. Logical extraction gets only the dump of existing files like call history, SMS or text messages and does not get the dump of unused space on the phone. Physical extraction gets the complete memory dump, but this is difficult to perform without causing damage to the original. Most of the forensic tools cannot bypass passcode of the smartphones[3]. Smartphones need to be jailbroken or rooted in order to access the data required for digital forensic investigation. Most of the mobile forensic solutions / products exist for the scenarios after the security incidents has occurred. Currently, there are no well-defined Digital Forensic Readiness (DFR) Framework for mobile computing devices (MCD). DFR data collection is still huge [58] [94]. What evidence to collect, what if the collected evidence is itself tampered [134] and lack of ability to monitor / target particular evidence. These are some of the challenges security professional, forensic investigators are grappling with. There is a need for new ways of identifying user before any security incident occurs and post the incident.

Tracking user’s search key words is one way search providers are gathering user preferences and display most appropriate ads targeted for respective users [149]. In case of a security incident happens where the user is suspected to have planned the crime with the help of internet searches, and has carried out some online purchase or ordering of object(s) or service(s) used in crime. In order to establish the correct sequence of events which were involved in the planning or execution of the crime, the knowledge of accused’s ads preferences can be helpful in the digital forensic investigation process. If the suspect had used mobile devices, the kind of ads that he clicked/viewed could reveal more information about the suspect, his motive etc. Analysis of ads could be used as one of the new techniques in digital forensic investigation process. Although the browser history can reveal more information, if the user has actually clicked on a particular ads and visited the website, this information reveals much more relevant and accurate information about user interests and behavior [22]. This is our motivation for considering user clicked ads as the source for identifying the user.

In similar work Toubiana et al. [136] have shown that Google web search session cookies can

expose an individual's personal data. They have claimed to recover around 80% of the user's search click history through these cookies. Castelluccia et al. [22] also show and demonstrate the leakage of a person's private information from the web searches.

Korolova [79] describes how the micro-targeted Ads leak an individual's private information on facebook. Castelluccia et al. [22] show that knowledge of only a small number of websites containing Google ads could reveal an individual's interests with an accuracy of 79%. The individual's Google ads profile can be reconstructed up to 58% using the same information. The authors claim to have accessed the ads, that are "almost always served in clear", by capturing respective person's data packets over the network [22].

Most of the current work has focused on the desktop web searches, browsing history, ads displayed on desktop browsers and have proved that person's private information gets leaked. In our work we develop a system to capture user urls from mobile device cache files, cookies and history database. Develop an algorithm to retrieve ad specific urls. Extract the privacy information that can be deduced from the ads. Develop a system for reconstructing the user behavior, sequence of events and the possibility of establishing user identity. We use this system to identify user across multiple networks, following different usage patterns on the same device or multiple devices. In this work we make following contributions:

1. Create a system which can track clicked ads live, extract these ads, analyze the ads for retrieving personal information and use this information to reconstruct user profile.
2. We conduct an experiment with multiple users, on multiple devices, capture over 5000 ads, analyze the ads and demonstrate our system.
3. We show that if a user is using his or her mobile at office environment with restricted network and using the same device at home environment with unrestricted network, following two different usage patterns we can still identify him or her to be the same user.
4. The developed system can be used as one of the approaches in digital forensic readiness framework for mobile devices.
5. This system and method has a wide range of applications in context based security, proactive and reactive digital forensic investigation.
6. The developed system can be used to identify any ads violating regional compliance.

The rest of the chapter is organized as follows. Section 6.2 provides the background for this work. Section 6.3 describes the methodology. Section 6.4 describes the implementation for the Mobile devices. Section 6.5 provides the results and inferences. Section 6.6 provides the chapter summary.

6.2 Background

In the following section, we present background information on various aspects discussed throughout this chapter: Types of mobile ads, mobile ad targeting and mobile ads architecture. There are various ways in which ads are displayed to the users on mobile devices such as SMS, MMS, call-only ads, ads within mobile web browsers and mobile applications. The various types of mobile ads [150] are described in the table 6.1. Different types of mobile ad targeting [61] are used by advertisers to target the users. The various types of mobile ad targeting are described in the table 6.2.

Table 6.1: Types of mobile ads

Ad Type	Description
Video Ads	Ads in between the running application or a game, user has the option to skip after certain time or watch
Interactive Ads	Same as Video Ads, in this case ad allows the user to interact with it
Banner Ads	Displayed within the app or site. The ads can displayed in any location like bottom centered, top centered, etc within the display limit based on the choice of placement of developers
Native Ads	The ad content is aligned with the content type of the app and does not feel like they are watching the advertisement
Pop-up and Takeover Ads	Show as a dedicated full screen page which user has to click to get to the original page
Lock-Screen Ads	Displayed everytime the device is locked
Notification Ads	Displayed as notifications from the apps which belong to a brand or company. They notify users of their happenings and in turn maintain and strengthen customer relationships
Rich Media Mobile Ads	Interactive ads which provide rich user experience using mobile device characteristics, like gyroscope, camera, accelerometer, etc
Branded Mobile Ads	Application specifically developed by an advertiser and uploaded to an app store

Table 6.2: Types of mobile ad targeting

Ad Targeting	Description
Content Targeting	Based on the app or the site on which ads get displayed
Behavioral Targeting	Based on the behavior of user on the device, user's browsing behavior, sites visited, recent downloads and also users interests by analyzing recent device location
Device or carrier based targeting	Based on type of device of carrier used. An ad for iPhone cases appears only on iPhone device
Demographic targeting	Based on information such age, gender, ethnicity and Language preference
Geographic targeting	Based on the user's location that is obtained from device GPS system or based on nearest cell tower location
Re-targeting	Based on the users who have viewed or clicked the ad in the past
Time Base targeting	Based on the particular time of day

6.2.1 What information does ads reveal

Each of these ads may contain information which can reveal some private information about the user which otherwise user would not disclose. This falls under unintended data leakage vulnerability through Ads [18]. Following are some of the user and device related information that gets leaked.

- Details about the app name and version on which ad was clicked
- List of device capabilities
- Name of the network operator
- User supplied age
- User provided gender
- Ad publisher account id
- Type of network being used like 3G, 4G or wifi
- User set system language
- App supplied keywords
- User location
- Current timezone
- Demography
- Emotional state of user (anger, fear, sadness, depression, hopelessness)

6.2.2 Mobile advertising architecture

Following diagram figure 6.1 explains the mobile advertising architecture. The workflow of the advertising system in mobile includes four participants: mobile applications, advertisers, ad exchanges and ad networks [82].

1. An Ad Agency manages marketing, advertising, and public relation services on behalf of advertiser. Ad agency handles overall marketing and branding strategies for its clients (Advertisers).
2. An Ad Network collects ad spaces from various publishers and segments it to offer specialized group of ads to match with the advertiser's demand

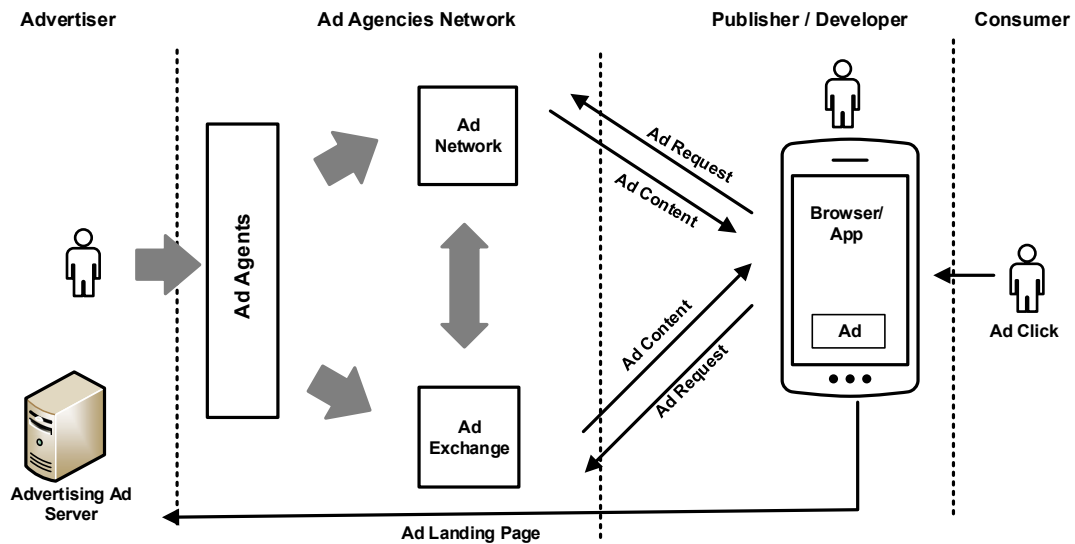


Figure 6.1: Mobile ads architecture

3. Ad networks is a market place between publishers and advertisers which aggregates ad inventory from different publishers and ad agencies or advertisers can buy them from single or multiple sites. The ad networks that participate in the exchange of their ads, place their bids on behalf of their customers, i.e., advertisers. The ad networks receive information about the user profile, context and device type from the ad exchange server. Ad exchange servers also shares the ad metrics info such as the number of clicks for an ad.
4. An Ad Exchange is also a market place between publishers and advertisers to buy and sell ad spaces. But this connects publishers with multiple ad networks. The buying and selling process in ad exchanges happens mostly through real time auction. Ad networks collect bulk ad impressions from these ad exchanges and resell them to advertisers. Ad exchanges are neutral parties that collect ads from different ad networks. The ad exchange server tracks down the list of displayed and clicked ads and determines how much money an advertiser has to pay and money that has to be passed to the developer of the app where the ad was displayed.
5. Publishers are developers of the mobile application or owners of the mobile websites who provide spaces within their app or web page for displaying ads. The mobile application includes an ad control, module that will notify the associated Ad Exchange server that there is an available ad slot on client's device. This app also sends user matrix such as profile, context, device type, etc to an ad exchange. It is now the Ad Exchange server that decides how to monetize the particular ad slot by displaying an ad.
6. Advertisers are one who wants to promote their products/services. The advertisers register with the ad networks through ad campaign. A campaign generally contains the advertising budget and the target number of impressions/clicks within a certain deadline.

6.3 Methodology

Ads could be of different types like location-based ads, content-based ads or targetted ads. Each of these ads contains information which can reveal some private information about the user which otherwise the user would not disclose. To capture and analyze the ads, we created an experimental setup described as follows:

- Targeted OS: Android, iOS
- Targeted browsers: Chrome, Safari
- Experiment conducted: Using 5 different apps belonging to popular app categories, 4 different devices, 4 different users and captured over 5498 ads
 - Simulation of the mobile ads ecosystem
 - Ad extraction
 - Ad Analysis
 - Inferences from the analysis (Reconstruct user identity)
 - Real world applications using the identity

6.4 Ads on Mobile devices

In case of mobile devices ads are displayed at two places, one is on the Apps itself and second is on the search browsers. We analyzed ads on iOS and Android OS.

6.4.1 iOS ads architecture

In case of iOS as explained in the figure 6.2, whenever user clicks on the ads on the Apps, the ads information gets stored in the cookies as shown in figure 6.4 and the cache files as shown in figure 6.3 under respective App folder and also an entry gets logged in the Safari History DB.

Whenever user clicks on the Ads on the Safari browser, an entry gets logged in the History DB. Ads information gets stored in the format as shown in 6.5. When user clicks on Ads either on the Apps or on the Safari browser, the safari history DB is common location for the ads information as shown in 6.5

iOS ads extraction

The ads were extracted by looking for ads tagged with <http://googleads.g.doubleclick.net>, <http://adclick.g.doubleclick.net> from safari history database. The in-app ads and the browser ads that are sent to browser were captured from browser history database file which contained

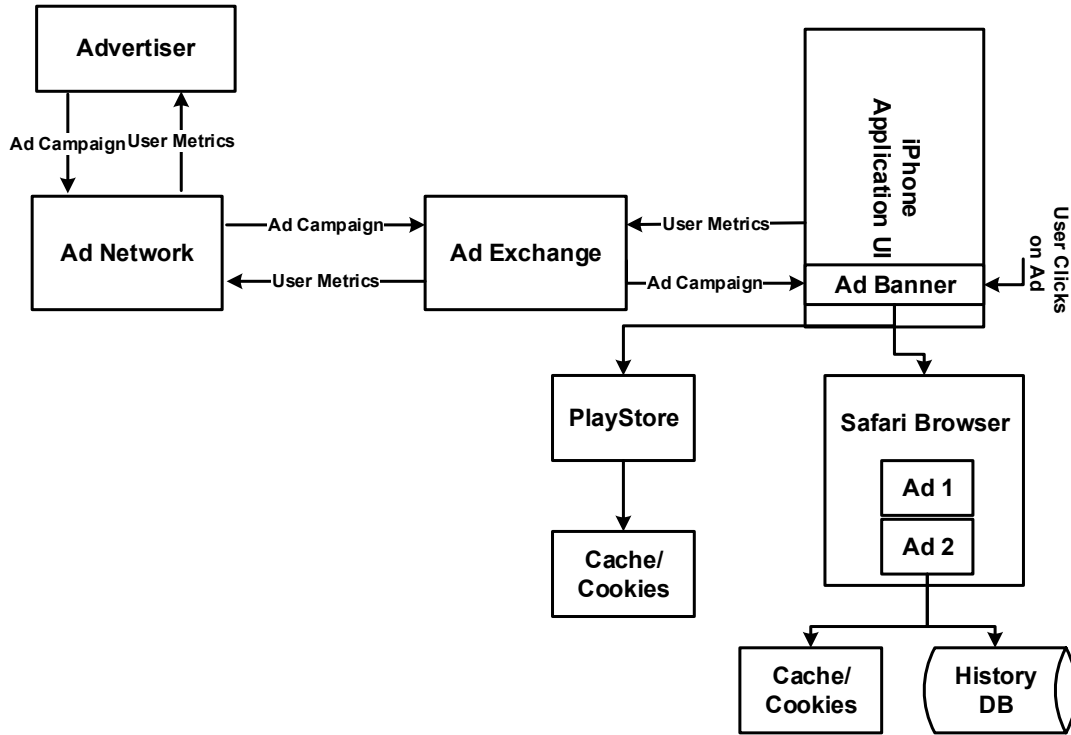


Figure 6.2: iOS ads architecture

ad url and also the timestamp. We were able to access the history database file after jailbreaking the iOS device. The ads urls were extracted from history database using keywords adurl, googleads, doubleclick. The ads sent to appstore had an intermediate link which get opened in browser and then redirected to appstore. These intermediate ad urls were again stored in history database, but the actual ad url was not stored. The intermediate urls were extracted from history database and then an app was developed to replay these intermediate urls to capture the actual ad url and other information related to ad. To capture all the ads in real-time, the iOS device had to be jailbroken and few of the SDK functions related to browser and playstore are to be hooked. The flow chart in the figure 6.6 explains the iOS ads extraction flowchart.

6.4.2 On Android

In case of Android as shown in the figure 6.10, whenever user clicks on the ads on the Apps, the ads information get stored in the logcat. Whenever user clicks on the ads on the chrome browser, an entry get logged to history database as shown in the figure 6.8. For some of the ads clicked on the app go to playstore, get logged to logcat and are accessible only on rooted devices. In summary, when user clicks on ads either on the apps or on the chrome browser, the logcat as shown in figure 6.9 and browser history database are two locations for the ads information.

request_key
ads.com/ads/preferences/css/static-styles.css[google.com]
ads.com/ads/preferences/css/mobile-styles.css[google.com]
ads.com/ads/preferences/js/mobile_ads_preferences.js[google.com]
www.google.com/css/gcs.css[google.com]
ads.com/ads/preferences/images/mobile_logo.png[google.com]
http://makeinindia.com/
adsservices.com/pagead/conversion.js[makeinindia.com]
wp-content/themes/Mil-Beta/lib/js/global.js[makeinindia.com]
wp-content/themes/Mil-Beta/lib/js/respond.js[makeinindia.com]
wp-content/themes/Mil-Beta/lib/js/picturefill.js[makeinindia.com]
wp-content/themes/Mil-Beta/lib/js/flexslider.css[makeinindia.com]
wp-content/themes/Mil-Beta/style-ver=3.9.2.css[makeinindia.com]
includes/js/jquery/jquery-migrate.min.js?ver=1.2.1[makeinindia.com]

Figure 6.3: Cache-ads information

Os3ywZvTviUzisyf6KW3eHhMKY6L0AMyz
dmley78Ljk6ltTcmKmhEvv6ilckMV_t-
NFdM 0AE3
UÍBj8GNP3Ö°A,Ö°A.checkmarx.com__hssc/
206289484.1.1428485271204\$8GNP%A,Ö°
A.checkmarx.com__hstc/
206289484.dc3c9426268d136b5846243fe
8c8c735.1428485271201.1428485271201
1428485271201.18GNP%A,Ö°A.checkmar
x.com__utma/
158400744.1016659334.1428485271.142
8485271.1428485271.1j8GNP3Ö°A,Ö°A.c
heckmarx.com__utmb/
158400744.2.10.1428485271R8GNPn.Ö°A
,Ö°A.checkmarx.com__utmt/
1D8GNPÖÄ»A,Ö°A.checkmarx.com__ut
mz/
158400744.1428485271.1.1.utmcsr=googl
e-ppc-utmclid=C1bG-
av5sOCFTllniod9hAVAL

Figure 6.4: Cookies-ads information

Android ads extraction

The flow chart in the figure 6.7 explains the Android ads extraction flowchart. The in-app ads or browser ads either open in the browser or in the playstore. We found that the in-app ads that get opened in the browser and the browser ads were getting stored in the history database. Now the challenge was to extract the history database and separate the ads url from other normal web browsing urls that are stored in database. For accessing the history database file of any browser, device has to be rooted. But we found that by using the APIs provided by Android SDK which are related to content providers of chrome and android default browser, we were able to extract the history file without actually rooting the device.

After extraction of history database, next task was to find a way to separate the ad urls from the normal web browsing urls. After analyzing the ad urls, we found out that the ads url have some unique keywords unlike other urls like googleadsservices, adclick, adurl etc. So these became the distinguishing feature for separating the ad urls from web browsing urls. Based on this we developed an 'Ad Extractor' android app which will log the date, time, title and url of all ad urls from browser history database to the logcat. We then created an algorithm to separate the ad urls based on specific keywords as mentioned above. The ad urls were separated from the other urls in the logcat and were copied to a file using ADB commands. Whenever 'Ad Extractor' app was launched it would dump all the ads data to a specified file. In this way ads from browser and ads from apps that go to browser were captured.

Next task was to capture those ads that go to play store from the app or browser. The urls of these ads are not captured in browser history file. We found that around 40% of ads that in android applications (in-app ads) go to playstore. These ad urls were not stored in browser

No Service 3:34 pm 100%

< Tables history_items

select rowid,* from 'history_items';

id	id	url
104	104	http://adclick.g.doubleclick.net/acik?sa=L&ai=BEoOyh8MwVfDIE9KTvgTwqYC4D-vttOp8A
104	104	http://m.idiva.com/
105	105	http://www.google.com/ads/preferences/html/mobile-abo
105	105	http://makeinindia.com/
106	106	http://googleads.g.doubleclick.net/acik?sa=L&ai=CMmQP-cMwVf-2Os6moAOA0oD4C-_s
106	106	http://googleads.g.doubleclick.net/acik?sa=L&ai=BoQfvM8QwVYP-EsGjoAPNkoGAC0aQ
107	107	http://hihonor.in/products/mobile/

Figure 6.5: History db ads information

history file. Hence the extraction algorithm used for ads that goes to browser was not applicable for extraction of ads that went to playstore as there was no playstore history database file from which information could be extracted. We found that the url scheme of playstore can be used in other user created app. Now while opening the app through urlscheme, it will be based on user's choice whether to go playstore or to use our new created URL schema app. There are two url schemas for playstore i.e. market:/ and http://play.google.com. So we developed an 'URL Schema' app with url schemes same as that of playstore. So now if we clicked on the ads that would normally go to playstore, Android device would show two options to the user:

1. The Playstore App (inbuilt app)
2. Our URL Schema App (customer built app)

On selecting our 'URL Schema' app the ads were opening in our app and the url of those ads were captured. We created two 'URL Schema' apps one that captures the ads with url schema market:/ and another with http://play.google.com. All of the above steps and apps were combined (browser ad extraction and playstore ad extraction algorithms) into a single app and deployed for volunteers. Whenever the users click on the ads, the app captures all the ads, stores in a temporary location and a mail is sent to authors based on predefined schedule.

In order to simulate the mobile ads ecosystem, an automated experimental lab set up using genymotion emulators was created. The ad clicking process was automated. For the automated process, we limited our focus to in-app banner ads (Ads that appear on the apps). We selected five apps for this experiment. The five apps were selected based on the popularity and covering

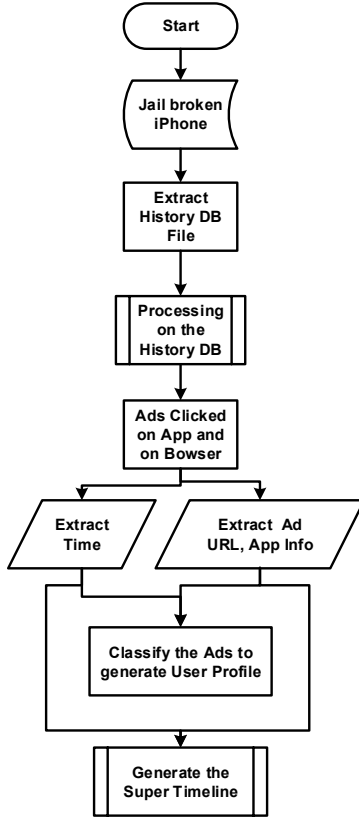


Figure 6.6: iOS-flowchart

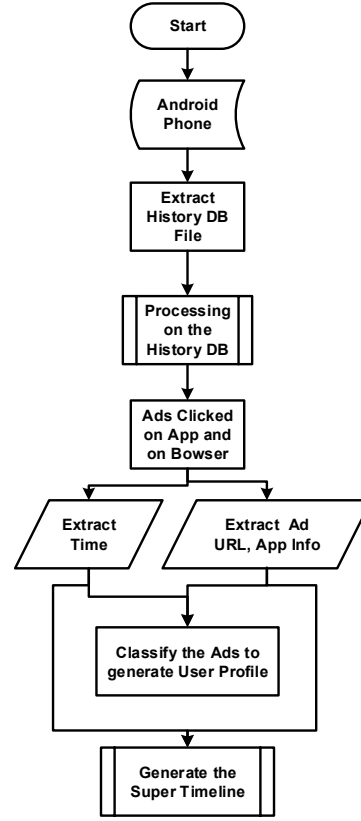


Figure 6.7: Android-flowchart

```

http://www.zomato.com/bangalore/tamarind-banaswadi?adref=043000-0050957-0000006&gclid=CJKhkZf9ubsCFWIT4godVWs
https://www.tapjoy.com/?gclid=CMr8k9ac6cQCFRQqjgodWyoApQ
http://www.askme.com/bangalore/taxi-cab-services?gclid=cn3o3o-p68qcfrcmjgodpieapq&utm_source=google&utm_medium=s
http://www.askme.com/bangalore/taxi-cab-services?gclid=cmzr-9dd68qcfesiqod23oalw&utm_source=google&utm_medium=s
http://lp.startapp.com/?mc=2&pid=3&cmpid=381&countid=WW&gclid=CMnr3sXe68QCFUQojgodKLwAWA
https://paytm.com/airtel-prepaid-mobile-online-recharge.htm?gclid=COCs2qT38sQCFQwnjgodgEwA3g
http://m.gaadi.com/used_car_result.php?campaign_type=search&make=Land+Rover&gclid=CJelv8qP88QCFRcMjgod-3AAGg
http://www.carwale.com/m/landrover-cars/?ltsrc=1123&gclid=CIou4tGP88QCFRcXjgod4hUAgg
http://www.webcrawler.com/info.wbcwrl.305.10/search/web?q=land+rovers+used+for+sale&cid=135636324&ad.network=g&a
  
```

Figure 6.8: Android history db

different categories like sports, news, games and a controller app. These apps had the banner ads at a fixed location which made the automated clicking more efficient. Following are the apps: Cricbuzz, Reddit_sync, 4pic 1word, Times of India and Advanced Permissions.

These above apps and two url schema apps were installed on genymotion emulators refer to table 6.3. A shell script was created to launch each of the above apps one by one and click on the ads. This process was repeated for around 1000 cycles. In our simulation, we capture the ads that go to browser directly from history database and for ads that goes to playstore, we capture the ads urls in url schema apps which are logged to a separate text file along with the timestamp.

```

Bangalore(UCA)&gclid=CMZ1rcXX-SQCFRUVjgodatQALQ
V/timeIds ( 8368): 16-4-2015 06:22:35
V/titleIds( 8368): Online Shopping India, Search and Buy Product Deals for Cheap
V/urlIds ( 8368): http://www.askmebazaar.com/product.php?app_data=cHJvZHVjdF9pZ
V/timeIds ( 8368): 17-4-2015 02:08:58
V/titleIds( 8368): New Cars, Used Cars, Car Prices, Reviews & Photos in India -
V/urlIds ( 8368): http://www.carwale.com/m/?ltsrc=l7831&gclid=C0n-tJn4_MQCFYEoj
V/timeIds ( 8368): 17-4-2015 04:08:06
V/titleIds( 8368): Idea Cellular - Cell Phone Services | 3G, Prepaid, Postpaid &
V/urlIds ( 8368): https://m.ideacellular.com/mobile/default/index.html?pn=insta
V/timeIds ( 8368): 17-4-2015 04:09:38
V/titleIds( 8368): Ford Figo Price in India, Photos & Review - CarWale
V/urlIds ( 8368): http://www.carwale.com/m/ford-cars/figo/?ltsrc=21545&gclid=C:

```

Figure 6.9: Android logcat

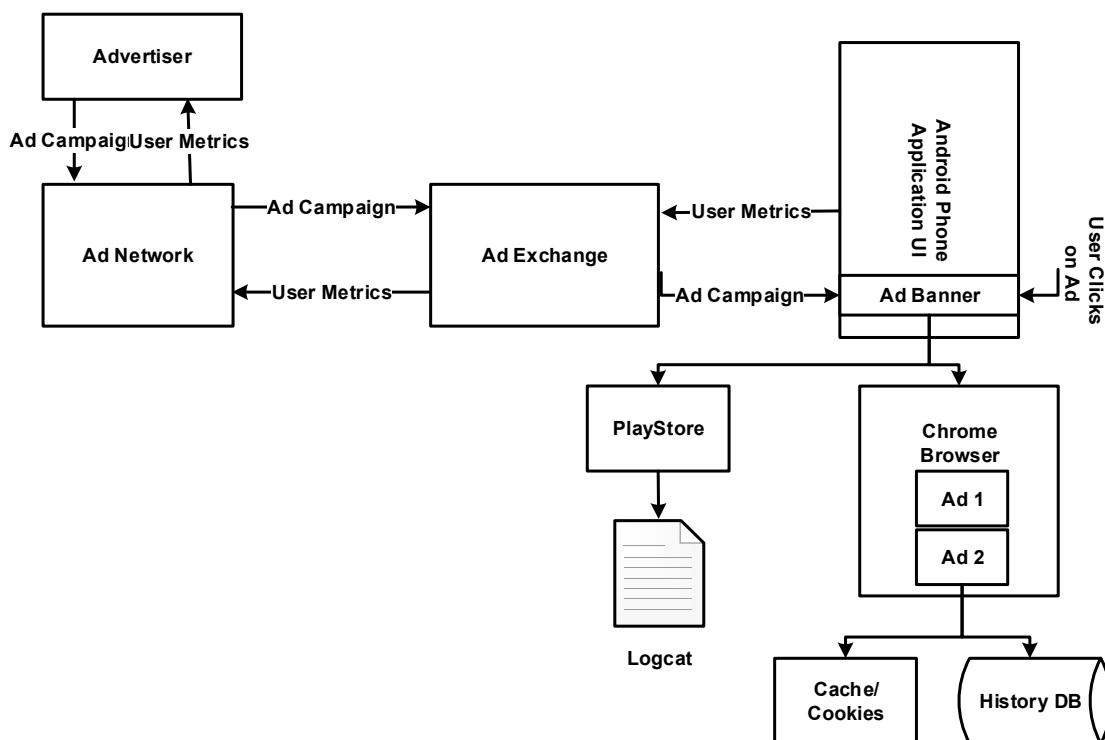


Figure 6.10: Android ads architecture

We performed this experiment with five different apps, two different users, running two emulators per user, over a period of eight weeks. Later the ads data was collected from history database and analyzed to construct user behavior profile. The extraction process is summarised in the figure 6.11

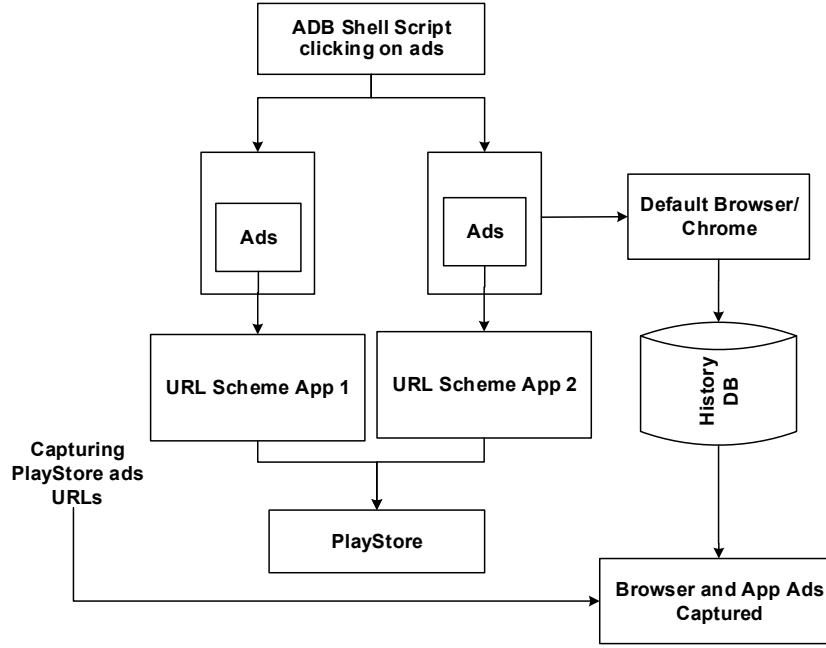


Figure 6.11: Android ads extraction experimental setup

Android ads analysis

Our experiment captured over 5498 ads from 4 different emulators refer table 6.3. We analyzed the data to mine for following data

1. Ad category Information - To understand different categories of Ads
2. Ad Interests Information - To understand what are the different types of user interests that Ads reveal
3. User's Information - To infer the user related information to construct his profile, generate user preferences based on the clicked Ads interests

For the experimental setup, we created two google accounts and created an automated program to read the URLs and to browse the links. We ran the following experiment, with two different use profiles and compared the results (refer to figure 6.16).

1. Logged-in as user1, started automated program to read the URLs and browse the links for one device in office environment (restricted network)
2. Logged-in as user1, started the automated program to read the URLs and browse the links for the same at home environment (unrestricted environment)

Table 6.3: List of Geny-motion emulators.

Devices	Description
Nexus(Machine 1)	Google nexus 4
Samsung(Machine 1)	Galaxy S3
Nexus(Machine 2)	Google nexus 4
Nexus(Machine 2)	Google nexus 4

6.5 Results and inferences

Our experiment captured over 5499 ads from 4 different emulators. Following are the statistics of the results captured:

6.5.1 Ads category

We found that 47% of ads belong to Business category, 20% Entertainment, 14% belonged to Productivity and 8% on News (refer to figure 6.12).

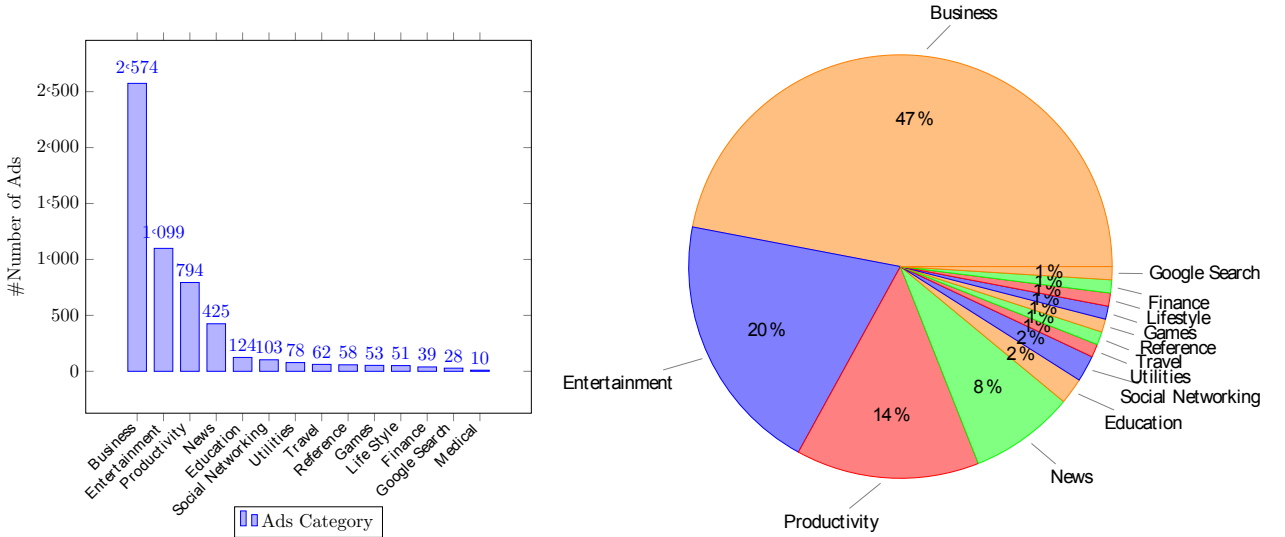


Figure 6.12: Ads category distribution

6.5.2 Ads interests

We found that 34% of ads belong to Real Estate, 20% Arts & Entertainment, 15% belonged to Internet & Telecom and 10% on Shopping (refer to figure 6.13).

6.5.3 User information

We found that 36% of ads contain location information, 33% Time information, 13% Language Information and 5% about the App based on the statistics. If we consider the ads clicked by the

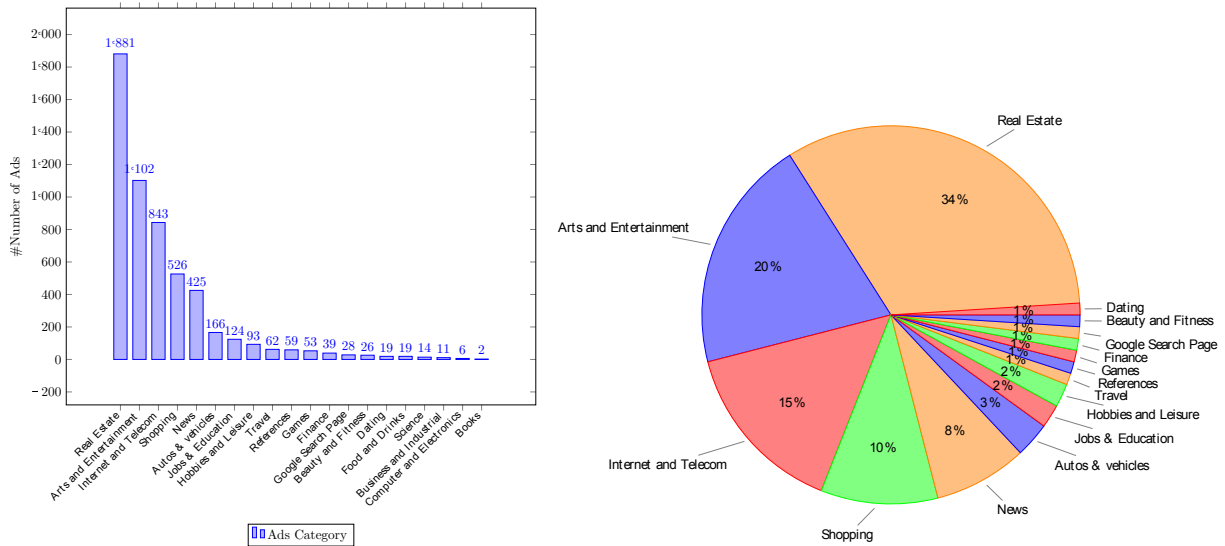


Figure 6.13: Ads interests distribution

user for more than 100 times, following are the statistics 38% contain the location information, 34% contain time information and 14% language information (refer to figure 6.14).

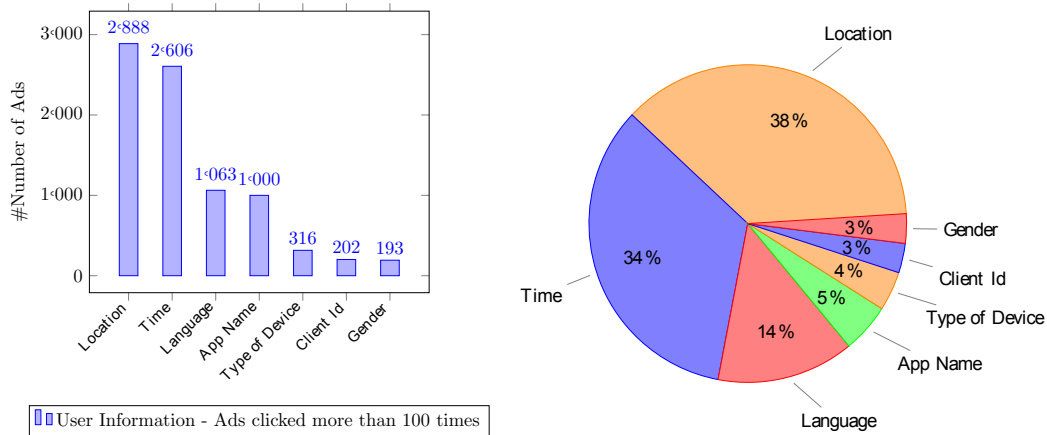


Figure 6.14: User information based on ads clicked more than 100 times

If we consider the ads clicked by the user for less than 100 times. Following are the statistics 22% about Financial transactions, 10% about some kind of offers, 10% about cars and 7% about Marital status (refer to figure 6.15).

Figure 6.16 shows the user preferences generated for the user using two different environments. Based on this profile we were able to link the user preferences belonged to the same user.

6.5.4 Inferences

Following are the inferences gathered from the data collected from ads that are displayed in Android applications.

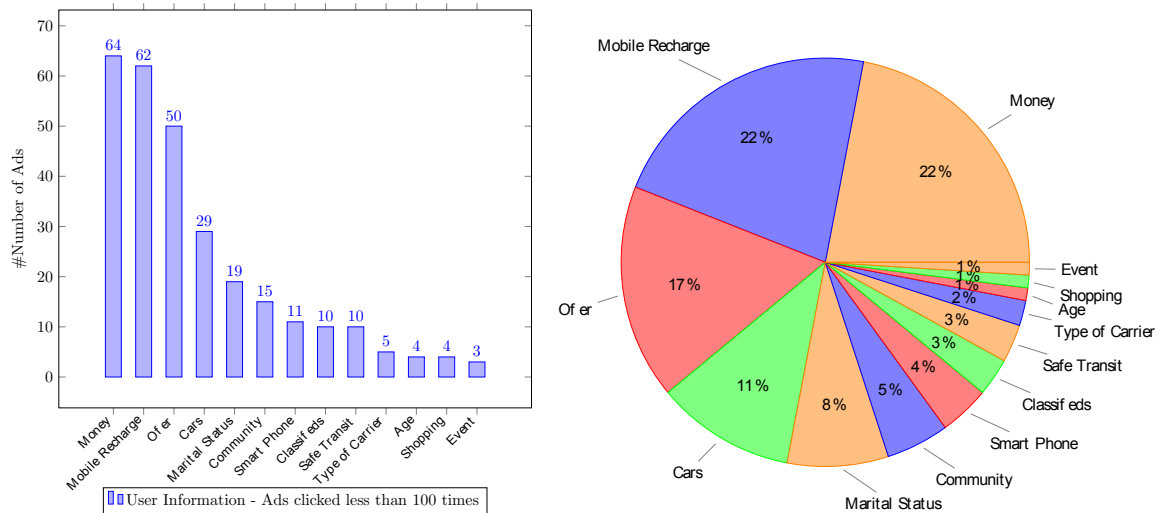


Figure 6.15: User information based on ads clicked less than 100 times

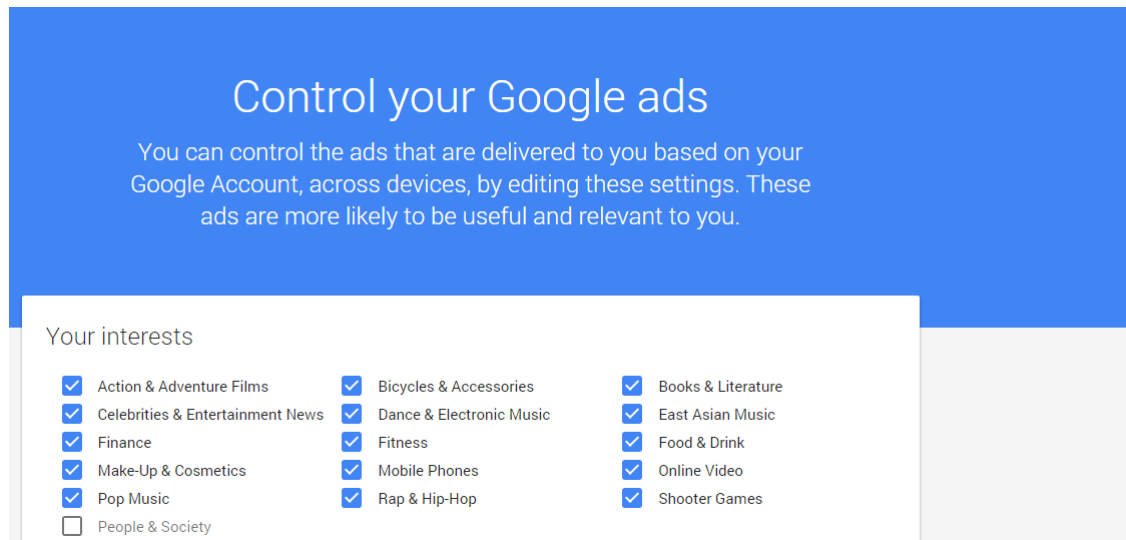


Figure 6.16: User preferences generated through Google

- Ads discloses information relevant to events, offers and occasions
 - Ex: - Offer in one of the Real Estate Ads which lasted for 5 days and we observed that the ads in these 5 days were related to that offer.
- Ads discloses information about user location Ex: - One of the Jeweler ad shows location and offers related to Bangalore, Ads related to regional language like Kannada news app.
- Ads discloses information about user gender, demography, login name, screen name, age group, account number, payment information, user preferences, user interests etc.
- There was no significant difference observed in displayed ads with respect to user logged in.

Following are the inferences gathered from the data collected from ads from web browser

- Ads on Google search
 - Discloses information relevant to regional festivals, events and occasions Ex: - Diwali, New Year, Christmas festival offers, sports events etc
 - Ads customized to geolocation, discloses information about user location Ex: - taxi-wala.com is the taxi service located in Delhi
 - Average database size of ads for a given keyword was around 10 with minimum being 5 and maximum of 10
 - Ads were almost always getting picked up from this database of ads
 - Percentage of ads displayed from the database varied between 30% to 100%
 - For some search keywords, more Ads seemed to appear late in the evening thus disclosing information about time of the day
 - Not significant difference observed in the ads displayed irrespective of whether logged-in or not logged-in...

From the results we were able to construct user profile, predict user location, date and time of day, regional festivals and events

6.6 Summary

In this chapter we described how Ads on a system can be used to identify an user. We created a system to collect these ads information and successfully demonstrated that user behavior profile could be constructed. Given the access to Mobile ads in any criminal investigation, forensic experts could use this information as one of the ways to construct the sequence of events and implicate the criminal without depending on the search providers for any kind of information. This is another system and method to authenticate or identify the user. Apart from application in Digital forensic investigation, this has its application in various areas

- Context based security - Creating a collective collaborative user behavioral pattern from Ads taking source urls from various devices, different accounts and using this pattern as one of the factors in two factor authentication.
- Predicting a potential security incident based on live ads profiling/forensics.
- Identifying an undesirable event based on the past history of similar incidents.
- Identifying the fake/malicious ads based on the deviation from the pattern of normal ad behavior.
- Authenticating the genuine user based on occasional ad profiling on his/her system.

- Identifying genuine active users and detecting/segregating a suspect based on active network tapping, provided we have the ad profiles of authorized users beforehand. Detecting Identity spoofing on a machine using Ad profile analysis.
- Identifying malicious use of company resources by active ad filtering.
- Continuous authentication of critical systems based on ad profiling of prominent users in corporate environment.
- Authenticating regular financial transactions based on the user's ads profile.
- Identifying any ads violating regional compliance. For example prenatal sex determination is banned in India, watching porn is illegal in certain countries. Ads violating law can be detected and alert raised.

Chapter 7

SYSTEM AND METHOD TO PROTECT SMART DEVICES APPLICATION BINARIES AGAINST CYBER ATTACKS

In the previous chapters, we discussed the digital forensic readiness system for mobile computing devices, forensic readiness and security solution at the application and operating system layer for iOS and Android, novel approach for analyzing mobile ads to identify user identity. In this chapter we propose a system and method to protect smart devices application binaries against cyber attacks.

Cyber attacks on smart devices (smartphone, wearables) exploit application vulnerabilities and carry out attacks like malware injection or runtime app manipulation. These attacks target the data available on smartphones and the paired wearables, either at rest or in transit. Therefore, securing the applications running on the smartphones as well as the paired up smart gadgets is critical to prevent these devices from getting compromised. Most of the present solutions advice following the Secure Development Lifecycle (SDL) to prevent vulnerabilities, followed by conducting security assessments to identify any vulnerabilities and then fixing them. Existing solutions are limited by the need to modify the source code, dependence on the developer's expertise and external libraries. In the current work, the authors have proposed a system, named 'MobiSecureWrap', which designs the best fix for the detected vulnerability in iOS and Android app binaries, then automatically creates a wrapper to protect the app binaries without the need to modify the underlying code. The authors have applied machine learning to recommend the best-fit resolution for the given vulnerability, and the solution has been assessed with over 5121 iOS and Android mobile apps achieving the prediction accuracy of 95.3%

7.1 Introduction

The smartphones, smartwatches, and other smart wearables are vulnerable to attacks at multiple layers like the OS layer, the hardware device layer, the infrastructure layer, and the app layer. As per Gartner report [47], 75% of mobile apps fail security analysis and attacks on such devices target the app layer vulnerabilities. Jailbreaking or rooting of these devices further weakens the OS level security features leaving it exposed to various attacks. Jailbreaking also opens up new entry points for cyber attacks on gadgets like smartwatches and other wearables that pair up with the smartphone. A malicious user or a hacker would perform jailbreaking [92] or rooting [132] on the target device before attempting to run the app in debugging mode, to perform reverse-engineering and dynamic hooking before gaining access to the application's source code and data.

The conventional approach for securing apps require following SDL practices [65] supported by various security assessments like SAST (Static Application Security Testing) and DAST (Dynamic Application Security Testing). The vulnerabilities identified during security assessments are later fixed by directly revising the source code, using external libraries like OWASP ESAPI [105]. The secure app development is expedited by utilizing static libraries [105]. Static libraries are designed to make it simpler for the developers to incorporate security in the source code. These libraries consist of APIs to validate user input, sanitize data, detect various types of attack payload and counter the attempt. The APIs could be invoked at proper places in the code to mitigate the vulnerabilities. However, drawbacks to this approach are the delayed delivery timeline, dependence on the skilled resources and dependence on third-party libraries. A delayed delivery timeline would mean the loss of business opportunities and increased exposure to cyber attacks. In case of a security breach, loss of brand reputation and market share.

Alouneh et al. [8] have focused on preventing buffer overflow attack by restricting access to string library functions and thereby protecting against overwriting. Their solution requires access to the assembly code which is then modified to include the protection code, and finally compiled to a binary executable. In our work, the proposed solution covers multiple vulnerabilities, works on the app's binary executable and injects the pre-built solution framework.

O'Sullivan, Pádraig, et al. [106] have incorporated security into an executable using binary rewriting. In their technique, they convert an x86 binary into an Intermediate Representation (IR). The IR is modified to include security functions for protection against buffer overflow, before getting converted back to the x86 code. In our work we do not need the IR, instead we load the dynamic library at the end of code segment stack which gets invoked whenever the application is loaded.

Appdome [10] provides a solution to add an essential security layer around the IPA/APK for iOS and Android apps, which protect the app data without any source code modification or SDK integration. This solution appears to be closest to our approach. However, Appdome's approach does not include machine learning and provide no trust mechanism for the user.

Patrick Wardle [141] in his blog talks about injecting dynamic library into the runtime process of the Mac OS. In our work, we propose inserting dynamic libraries to the apps which are app-store ready.

Diquet Alban et al. [4] provide an easy to integrate SSL pinning dynamic library for the iOS apps, which incorporate SSL pinning feature into the app without doing any code changes. However, their solution still requires recompilation of the code. Although it provides an SSL pinning API which protects the app from 'Insecure Transport Layer' vulnerability, hooking is still possible on the jailbroken iOS devices which would fail the solution by exposing the app to a man-in-the-middle attack (MITM).

Simone Margaritelli [86], in his blog, talks about injecting shared libraries into the Android apps at runtime. In our solution for iOS apps, we use the concept of injecting dynamic libraries, and for Android apps we inject a custom .smali code to safeguard the app from security threats. The injection into the app binary is accomplished without the need for source code modifications.

With the security API approach [70], the developer needs to integrate specific library(s) within the app; provide a reference to the library headers and other resources in the app project; make the required modifications to the source code by calling these APIs, and finally compile the code to get the end product. The above-stated process requires a good understanding of the library APIs as well as the integration procedure.

In all of the currently known approaches, following are the technical limitations and challenges.

- The app cannot be secured without modifying the source code followed by a recompilation
- No approach assures that the source code changes made could be trusted
- Increased app development lifecycle timeline
- Risk of cyber-attacks
- Increased time and effort to resolve security threats
- Inadequate skilled resources for security assurance
- Incomplete testing

- The above-stated processes consume much time which could delay the product releases
- As per the best of the author's knowledge, presently there are not many solutions which explore machine learning capabilities to propose security solution for any given vulnerability

These challenges and the limitations are the motivation for our research work. In this work, the authors attempt to address these shortcomings. We have designed and developed the non-intrusive methodology that could detect security vulnerability, recommend the most suitable fix, build a solution from the list of pre-built fixes and automatically fix the vulnerability. This solution protects smartphone as well as smart gadget application binaries. With the proposed system, the authors have made the following contributions:

1. An approach for creating a new wrapper to protect the existing apps, immaterial of their features.
2. Internally wrapping the apps in a secure envelope to translate them into a new digital binary app without modifying their original functionalities.
3. Keeping the app features intact, while the shielding process is carried out with well-defined protocols with no code variations thereby reducing the risk of cyber-attacks.
4. A reduced SDL time for the app, delivering faster time to market.
5. Introduced a machine learning system to recommend the solutions, consequently enhancing the efficiency of auto fixing.
6. Demonstrated a working prototype by evaluating the system with 5121 mobile apps.

The rest of this chapter is organized as follows: Section 7.2 describes the proposed solution, Section 7.3 describes the implementation details, Section 7.4 discusses the developed prototype, Section 7.5 provides the results and Section 7.6 discusses the conclusion.

7.2 Proposed solution

The solution approach consists of four main modules, namely the Trusted Dynamic Wrapper (TDW) Creation System, the Auto Wrapper, Machine learning module and the Blockchain for Trust module. Refer to figure 7.1) for the 'MobiSecureWrap' proposed architecture. Refer to figure 7.2) for the 'MobiSecureWrap' sequence flow diagram.

7.2.1 Trusted dynamic wrapper creation system

This system takes the app parameters like the bundle Id, package name which is a unique identifier for the app and creates a dynamic wrapper to work with the app. The dynamic

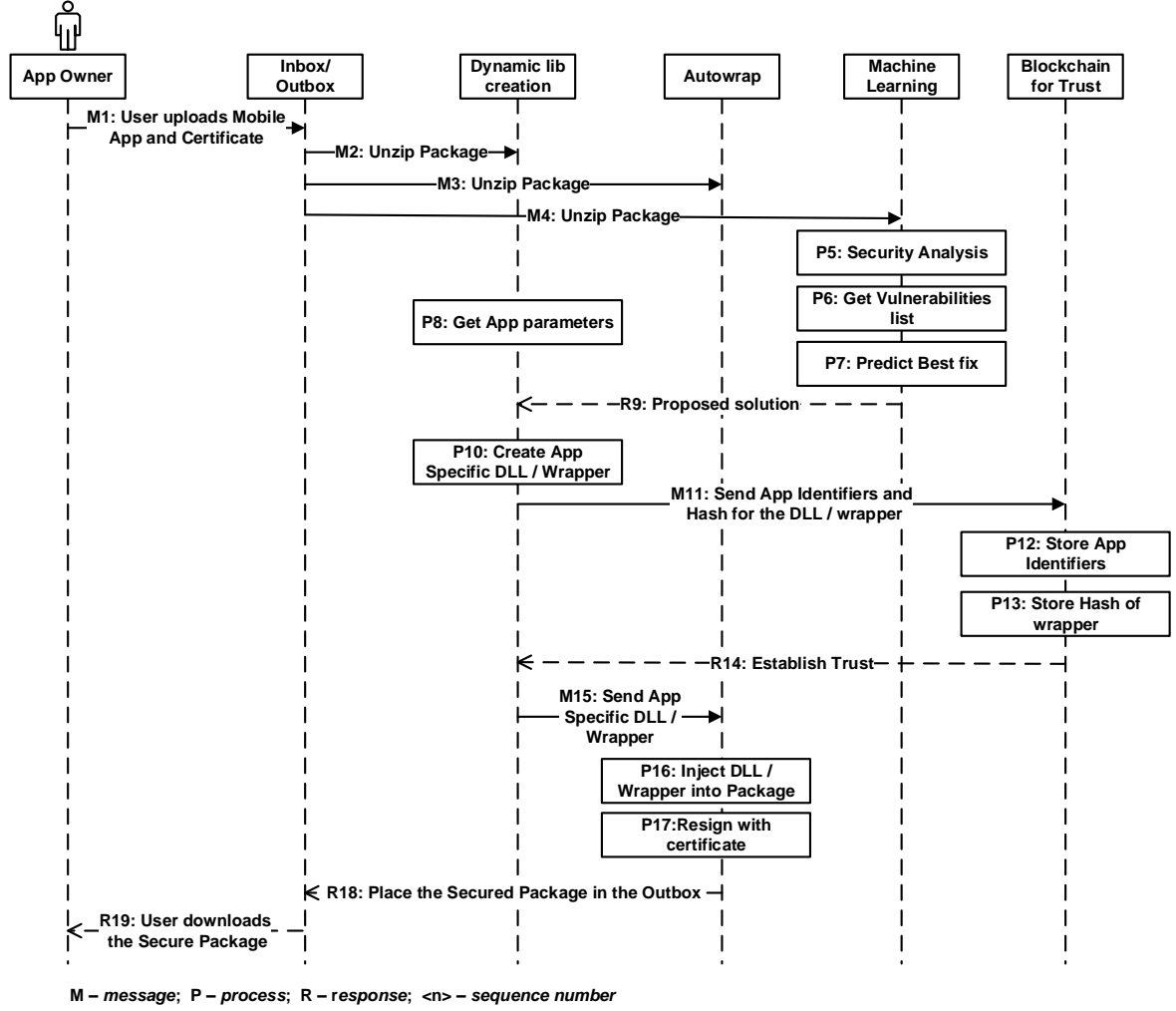


Figure 7.2: MobiSecureWrap sequence flow diagram

solution saves developer's time and effort by automating the entire process of integrating security into an already developed app. The developers do not have to understand any API-usage or perform any source code modifications. The solution does not require source code changes, does not call for recompilation, there is no need for any specific static libraries integration or for invoking any API calls during the development stage. This simplified process of injecting the dynamic library for iOS app or the smali code for Android app into the app-binary can happen at the production stage. This method facilitates faster deployment of secure apps to the marketplace. Figure 7.2 provides the sequence flow diagram of the proposed MobiSecureWrap system.

7.2.3 Machine learning module

In this work, we utilized the security analysis records database obtained from one of the author's previous research [52]. The database consists of vulnerabilities list for over 5000+ mobile appli-

cations. We choose the Naive Bayes algorithm (NBA) for classifying the apps based on whether the fix is present or absent. An app can have more than one fix for a given vulnerability. Hence one vs. the rest method for multi-label classification was employed. We choose NBA over other algorithms for the following reasons [23]: training dataset is relatively smaller, consists of a large number of predictors, requirement for scalable architecture, need for the faster algorithm and also need for multi-classification. The machine learning model was created based on the vulnerabilities found and corresponding mitigations for the applications. This module then takes the security analysis (vulnerabilities list) for the app to be secured, apply the learning model, recommend the best fix for the vulnerabilities and provide the solutions list to the dynamic wrapper creation module.

7.2.4 Blockchain for Trust module

This module takes the app identifiers such as bundle Id, package name which can uniquely identify the application, get the hash value of the dynamic dll / wrapper created by the dynamic wrapper module and stores this information as a block in the blockchain distributed ledger [85]. This process helps the service providers, app developers, app owners and app users to verify the authenticity of the wrapper.

7.3 Implementation details

In this research work, the authors have implemented the proposed solution for Android and iOS applications (phones and wearables).

7.3.1 Dynamic Wrapper creation and Autowrapping

iOS and watchOS apps (.ipa) The iOS applications are packaged in a bundle that consists of resources related to a particular app, namely the actual app-binary, images, different files like plist, user-interface files (viz: XML Interface Builder (XIBs), storyboards, NeXT Interface Builder (NIBs)), developer certificates, entitlement files, and more. The bundle is zipped with an extension ‘.ipa’, after which the IPA files can be uploaded to the app-store or installed directly on an iPhone. The dynamic libraries (dylibs) have backward support until the iOS version 8.0 and are called embedded frameworks. These dylibs, which are placed in the app bundle (inside the folder /Frameworks/), are loaded into the app at runtime.

The app-binary of iOS is in Mach-O format (Mach object file) [151], which is an executable format specific to systems based on MACH kernels, like the OS X, iOS, and the NEXTSTEP. A single Mach-O executable can be used to store multiple-architecture instruction sets. The iOS Mach-O binaries contain ARMv6, ARMv7, ARMv7s and ARMv8 architecture instructions.

Each Mach-O file in iOS consists of a header followed by a set of LOAD command section, which is accompanied by one or more segments like TEXT and DATA. The LOAD command section contains the symbol-table, the dynamic symbol-table, and a series of ‘LC_LOAD_DYLIB’ commands for successfully loading different dylibs. The figure 7.3 shows the Mach-O binary view of an iOS app before it is injected with a dylib.

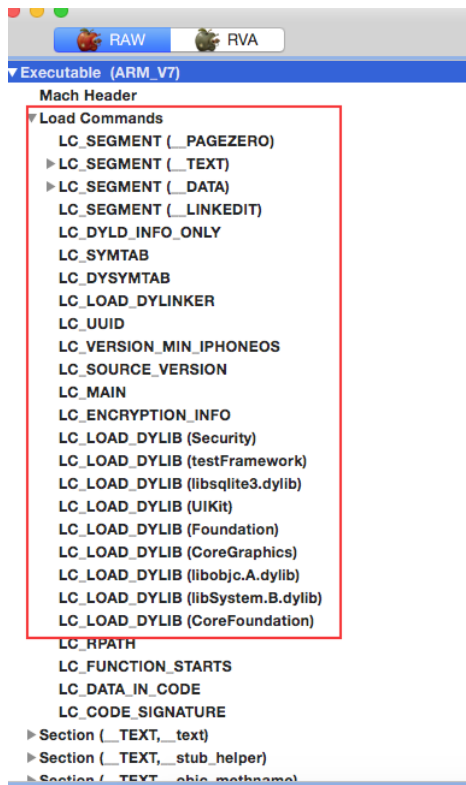


Figure 7.3: iOS - Before injecting *dylib*

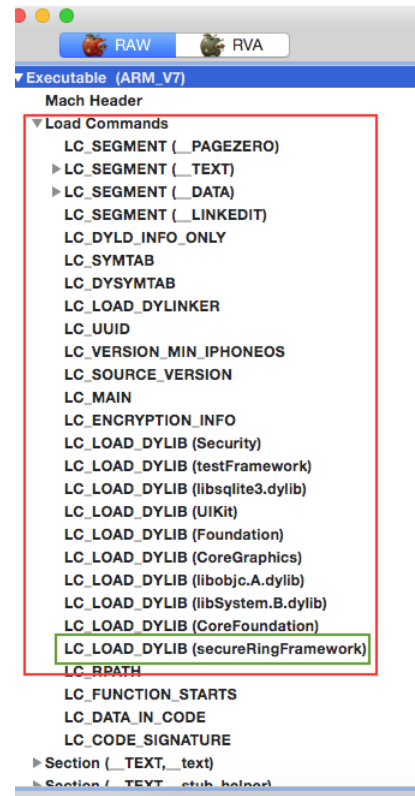


Figure 7.4: iOS - After injecting *secureRingFramework*

The proposed solution creates an iOS framework with dylib, using the Xcode Integrated Development Environment (IDE). The framework's dylib contains code for jailbreak detection, debugger detection, encryption check, and SSL API-hooking check in its constructor [50]. The framework's dylib code gets executed only when the app starts and after the library is loaded. The framework is added to the app bundle's path in the folder (/Frameworks/). Afterward, the app's Mach-O binary is modified to include an ‘LC_LOAD_DYLIB’ command which refers to the proposed dylib. The figure 7.4 shows the Mach-O binary view of the iOS app after the secureRingFramework dylib is injected. After the dylib is added and its binary is modified (for loading), the app-bundle needs to be re-signed with a legitimate certificate before it could be repackaged into the ‘.ipa’ file that is ready to be deployed on the app-store. The solution for jailbreak/root detection, runtime app-manipulation detection for SSL and other critical functions is implemented as explained in the previous section.

The solution for lack of binary protection from top ten OWASP mobile vulnerabilities is implemented as follows. The iOS apps use objective-c and swift languages which are dynamic programming languages. The iOS apps are built on objective-c runtime libraries which are used for dynamic binding of symbols at runtime. In the proposed solution, the authors have used this feature to implement the solution for vulnerabilities fixing. To implement the rebinding of symbols dynamically for selected methods in Mach-O binaries, the authors have used open source library called ‘Fishhook’ [42].

The Fishhook is based on ‘DYLD_INTERPOSE’ feature available for Mac OS X, which enables it to replace the pointers to the symbols in the runtime symbol table with the replacement method pointer for the methods/symbols. These symbols are exported from ‘iOS SDK dylib_shared_cache’ (all the system libraries on iOS have been combined into one big cache) or third party dynamic library/framework.

This approach is supported for all apps till iOS 8. However, the procedure does not work for iOS 9 and above versions. It fails on iOS 9 [32] because the code within ‘dylib_shared_cache’ will directly jump to the respective functions/methods without looking for pointers in the symbol table and hence any changes made to the runtime symbol table will not affect the methods that are called from ‘dylib_shared_cache’.

To address this challenge, the authors have adopted ‘Method_Swizzling’ [88] technique for implementing solutions for vulnerabilities like Insecure Data Storage, Insecure Transport Layer Protection, Unintended Data Leakage, and Broken Cryptography. ‘Method_Swizzling’ is a process of swapping the implementation of two methods, i.e., it updates the function pointers of the methods of a class. As discussed earlier, objective-c is a language with dynamic binding and hence supports the ‘method_swizzling’ technique.

In the proposed solution, the authors have implemented the methods for existing APIs within iOS SDK by adding techniques to mitigate the vulnerabilities and then swizzle the iOS SDK framework APIs with authors’ implemented methods. For example, to mitigate insecure data storage, one of the techniques is to encrypt the data stored in plist files. The authors implemented the methods for storing the data on plist files using encryption, and the original methods are replaced with authors’ implemented methods using ‘method_swizzling’.

Android and Android-Watch apps (.apk) The Android app is packaged as an Android Package Kit (APK). The APKs are used for distribution and installation as mobile apps for android based devices. An APK file consists of the program’s code, which includes a manifest file, application resources, certificates, and assets that are required for the application to run. The APK is an archive file similar to JAR file format with .apk as their file extension. The manifest

Name	Type
BuildConfig.smali	SMALI File
MainActivity.smali	SMALI File
R\$anim.smali	SMALI File
R\$attr.smali	SMALI File
R\$bool.smali	SMALI File
R\$color.smali	SMALI File
R\$dimen.smali	SMALI File
R\$drawable.smali	SMALI File
R\$id.smali	SMALI File
R\$integer.smali	SMALI File
R\$layout.smali	SMALI File
R\$menu.smali	SMALI File
R\$mipmap.smali	SMALI File
R\$string.smali	SMALI File
R\$style.smali	SMALI File
R\$styleable.smali	SMALI File
R.smali	SMALI File

Figure 7.5: Android - Original Smali files

Name	Type
BuildConfig.smali	SMALI File
DroidRingActivity.smali	SMALI File
MainActivity.smali	SMALI File
R\$anim.smali	SMALI File
R\$attr.smali	SMALI File
R\$bool.smali	SMALI File
R\$color.smali	SMALI File
R\$dimen.smali	SMALI File
R\$drawable.smali	SMALI File
R\$id.smali	SMALI File
R\$integer.smali	SMALI File
R\$layout.smali	SMALI File
R\$menu.smali	SMALI File
R\$mipmap.smali	SMALI File
R\$string.smali	SMALI File
R\$style.smali	SMALI File
R\$styleable.smali	SMALI File
R.smali	SMALI File

Figure 7.6: Android - New Smali file added

file contains information about various components of the respective app, like activities, services, broadcast receivers, intent messages, and more. These details help the Android system to know the components and their usage during the app's launch.

When an Android app's APK is decompiled using open source tools like ApkTool [146], it produces a manifest file, a smali folder and other components of the app. The java source files in the android app are decompiled as '.smali' files within the smali folder. These '.smali' files are the actual source code of the app. The figure 7.5 shows the original '.smali' files.

The 'MobiSecureWrap' solution creates one '.smali' file, for the target app. The '.smali' file contains an activity that is included with resolutions for the given vulnerability (viz: android root detection and mitigation for other vulnerabilities) as recommended by the machine learning module. The activity created by 'MobiSecureWrap' is designated as the main activity in the app's manifest file after associating the original main activity with an Intent filter. The newly introduced activity assumes the responsibility of the main activity, which ensures that the resolutions such root detection code are invoked before the actual app launch. After completing all the implemented vulnerability checks, it then starts the original main activity of the app with the aid of the Intent; ensuring the actual functionality of the app is protected.

The new '.smali' file (with MobiSecureWrap's activity) is copied to the Smali folder of the decompiled app (figure 7.6). The manifest file is modified as described above and the app is repackaged with the help of ApkTool [146]. Afterward, the re-packaged '.apk' file is signed with legitimate certificates ready to be uploaded to the Play-store. The figure 7.7 shows the updated manifest file.


```

<?xml version="1.0" encoding="utf-8" standalone="no"?>
<manifest
xmlns:android="http://schemas.android.com/apk/res/android"
package="com.test.sampletoaster" platformBuildVersionCode="23"
platformBuildVersionName="6.0-2438415">
  <application android:allowBackup="true"
android:debuggable="true" android:icon="@mipmap/ic_launcher"
android:label="@string/app_name"
android:theme="@style/AppTheme">
    <activity
android:name="com.test.aCart.DroidRingActivity"
android:label="@string/app_name">
      <intent-filter>
        <action
android:name="android.intent.action.MAIN" />
          <category
android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
      </activity>
    <activity
android:name="com.test.aCart.MainActivity"
android:label="@string/app_name">
      <intent-filter>
        <action android:name="DroidRingIntent" />
      </intent-filter>
    </activity>
  </application>
</manifest>

```

Figure 7.7: Android - Modified manifest file

7.3.2 Machine learning implementation

The implementation consisted of the three program modules, refer to the machine learning flowchart shown in figure 7.8

Creation of training data set: The training dataset creation module was developed using a Java program. The dataset was created with vulnerabilities and solutions records of about 5121 Android and iOS applications. The applications belonged to various categories like social networking, news, lifestyle, travel, and shopping (refer to table 7.2). The vulnerabilities for each of the application was used as the feature set in the training data. Predictors are the vulnerabilities and responses are the solutions to mitigate these vulnerabilities.

Test data set: Test dataset creation for each new app to be analyzed was developed with another Java program. This test data with the list of vulnerabilities is then provided as input to each of the models for predicting possible solutions.

Generating the Learning Model: Learning model was created used the R program. The authors employed the Naive Bayes algorithm [77] for classifying the apps based on whether a respective solution is present or not. The algorithm calculates probabilities for each category of the possible solutions; then assigns the solutions with the highest probability. The vulnerabilities for an app could have more than one solutions, therefore the multi-label classification, one vs. the rest approach [81] was chosen. In multi-label classification, a model is generated for each of the possible solutions. For instance, if there is a solution ‘Use encryption’, we use a binary classification with two classes: 0 and 1. 0 indicates ‘the solution is absent’ and 1 indicates ‘the solution is present’.

Thus, a model is generated for each of the possible solutions. The learning model can now be used to predict solutions for any new app with a given set of vulnerabilities. After the solution is recommended, the record with list vulnerabilities and the corresponding predicted solutions for the analyzed app is updated back into the training dataset to improve the accuracy of the system.

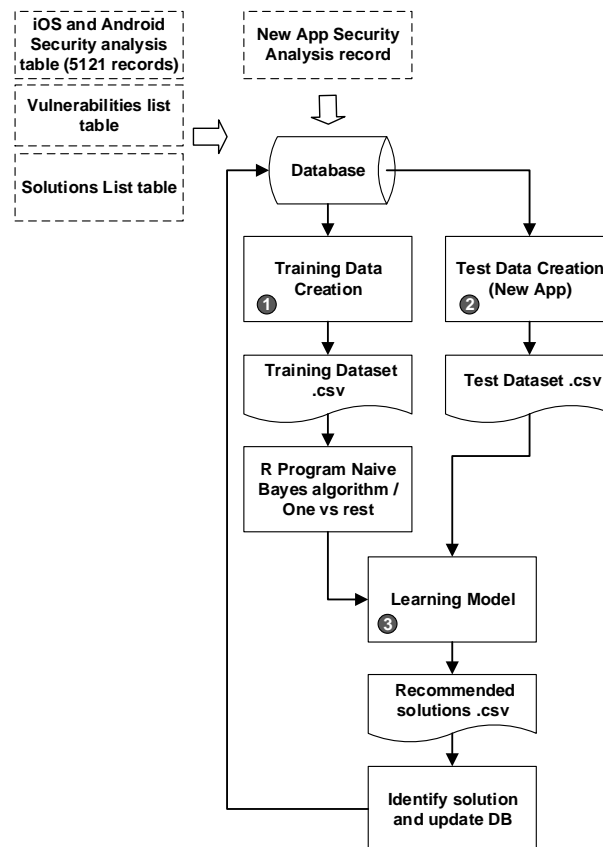


Figure 7.8: Machine learning flowchart

Training dataset creation

Inputs: Security analysis records containing the list of vulnerabilities and solutions.

Vulnerabilities: Vulnerabilities list is stored as table in the database. Refer to table 7.1.

Mitigations: Security mitigations [104] is stored as another table in the database. Refer to table 7.4.

Method: The program is executed only once for creating the training dataset. If the vulnerability is present then a '1' is updated, else a '0' is stored. The solutions are updated based on the mitigations applied for resolving the vulnerabilities. Refer to table 7.2, the columns V1 to Vn refers to the 'n' vulnerabilities in the app, and the columns S1 to Sn refers to the solutions

Table 7.1: Vulnerabilities list

Vulnerabilities
LACK OF BINARY PROTECTION
INSECURE DATA STORAGE
INSUFFICIENT TRANSPORT LAYER
PROTECTION UNINTENDED DATA LEAKAGE
BROKEN CRYPTOGRAPHY
CLIENT SIDE INJECTION
POOR AUTHENTICATION AND AUTHORIZATION
UNCHECKED RETURN VALUE
BUFFER OVERFLOW
FILE REFERENCED BY NAME
FORMAT STRING
INSECURE TRANSPORT LAYER PROTECTION
INCORRECT INITIALIZATION
WEAK CRYPTOGRAPHIC HASH
...
UNCHECKED CSTRING CONVERSION
MISSING CHECK AGAINST NULL
REDUNDANT NULL CHECK
LOGGING WEAK SERVER SIDE CONTROLS
INSECURE TRANSPORT: HTTP GET
INSUFFICIENT TRANSPORT LAYER PROTECTION
CUT AND PASTE LEAKAGE
UNINTENDED DATA LEAKAGE
SCREEN CACHING
SECURITY DECISIONS VIA UNTRUSTED INPUTS
CWE-259: HARDCODED PASSWORD
IMPROPER IMPLEMENTATION OF NSSECURECODING
BUFFER OVERFLOW INTERNAL
IMPROPER SESSION HANDLING – COOKIES
BUFFER OVERFLOW FORMAT STRING
INSECURE STORAGE: LACKING DATA PROTECTION
SIDE CHANNEL DATA LEAKAGE.

Output: The output is TraningData.csv file that consists of a solution list along with a set of vulnerabilities (keywords) for each app record (refer to table 7.2).

Test dataset creation

Inputs: List of vulnerabilities found in the new application.

Method: Reads the new app security analysis data from the database and checks for each vulnerability (V1 to Vn) from the vulnerabilities table in the database (Refer to table 7.1). If the keyword (vulnerability) is present, then a '1' is updated. Otherwise, a '0' is saved. The output is the 'TestData.csv' file with the list of all the vulnerabilities and solutions for the app. The list of solutions (S1 to Sn) is initially marked as 0 (refer to table 7.3 for sample Test dataset).

Table 7.2: Training dataset

AppName	Vulnerabilities				Solutions			
	V1	V2	...	Vn	S1	S2	...	Sn
aLogCat	1	1	1	1	1	1	0	0
Amazon Shop	1	1	1	1	1	0	0	0
Angry Birds	1	0	0	0	1	1	0	0
Asphalt	1	1	1	0	1	1	0	0
...			
BookMyShow	1	0	0	1	0	0	1	0
BOX8	0	1	0	1	0	1	0	0
BusyBox	0	0	1	1	0	0	0	0
...			
Callapp caller ID	0	1	1	1	1	0	1	0
Calorie Counter	1	1	1	1	1	1	0	0
Cam Scanner	0	1	0	1	0	1	0	0
Canara Enfobook	0	0	0	0	0	0	1	1
Candy Crush	0	1	1	0	1	1	0	0
CirclePay	0	0	0	0	0	1	0	0
Clash of Clans	0	0	1	0	0	0	0	0
CloudCheckServ	1	1	1	0	0	0	1	0
...			
DB Summit	0	1	1	1	0	1	0	0
Dominos	1	0	1	1	1	0	1	1
Download Mgr	0	1	1	0	1	0	0	0
...			
English Grammar	0	1	0	1	1	1	0	0
eRail	1	1	0	0	0	1	0	0
ESExplorer	0	0	1	1	1	0	1	1
...			
Facebook	0	1	1	1	0	1	0	0
Flipkart	1	1	1	0	1	0	1	0
FolderSync	1	0	1	1	1	0	1	1
Foodpanda app	1	1	0	1	1	1	1	0
...			
ZipRecruiter	1	1	1	0	1	1	0	0

Machine learning model creation - Java-R integration

The program calls and executes the R code from the Java program for generating the Learning Model.

R program: The program uses the training data, and the test data as inputs applies the Naive Bayes Algorithm. The resulting prediction model from the algorithm is applied to the test data. The recommended values from the model are stored in a file which can be read by the Java program. The training data is updated with the aid of the test data and the recommended solutions. The Naive Bayes algorithm uses the Bayes theorem for finding the probabilities:

$$p(Ck|x) = \frac{p(Ck)P(x|Ck)}{P(x)} \quad (7.1)$$

Where $p(Ck)$ is the probability of the presence a solution in the complete dataset. $P(x|Ck)$ is the probability of a particular instance when its solution is given. $P(x)$ is the probability of

Table 7.3: Test dataset

Solutions						Vulnerabilities				
S1	S2	...	S8	S9	S _n	V1	V2	V3	...	V _n
0	0	...	0	0	0	0	1	1	...	1

an instance when the complete dataset is given. $P(Ck|x)$ is the probability of the presence of a particular solution for a given app. We find $P(Ck|x)$ for each of the solutions. The recommended solution will be the one which has the highest probability.

A simple implementation of Naive Bayes can do only multiclass classification, but an app can have more than one solution, so we applied ‘One vs. the rest’ method for multi-category classification. A model is generated for each of the solutions.

Output: The predicted values for each of the solutions are exported into a comma-separated file. If a solution has ‘0’ value, then it is not present. The solution which has value as ‘1’ denotes its presence.

7.4 Prototype Tool

The authors have created a tool called ‘MobiSecureWrap SecureRing’. Any user can upload the Android, iOS or wearable app’s binary (IPAs or APKs), along with the certificate. The certificate is required to sign the app after the necessary modifications are complete. The figure 7.9 shows the MobiSecureWrap tool.

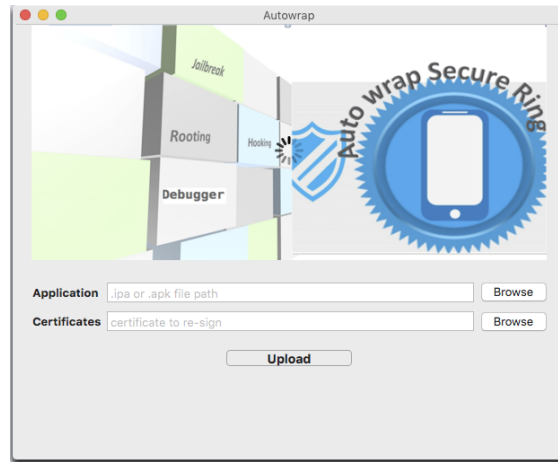


Figure 7.9: Uploading the app and certificate

The process of securing an app starts with a click on the ‘Wrap’ button. The app is first decompiled; the app parameters like bundle id for iOS and package name for Android are

Table 7.4: Solutions list

Security solutions
Detect app running on the jailbroken rooted device
Detect app running in debug mode
Detect app encryption is broken
Check runtime manipulation of SSL validation APIs
Use strong encryption
Local integrity checks to detect any unauthorized changes
Avoid deprecated algorithm like RC4, MD5, MD4, & SHA1
Disable auto correct functionality
Encrypt tokens in transit
User input validation
Use file encryption
...
Protect with device unlock code and delete on remote kill
Use SQLcipher for SQL data Encryption
Store in tamper-proof secure area
Context-based access (location, IP)
Scheduled deletion of historical tracking data
Remote wipe and kill switch
App-specific data kill switch
Use of secure element (Encrypted SD card)
Use repeated patterns
Use certificates signed provided by a trusted CA provider
Validate password entropy
Disable storing passwords keys in cache or logs
Detect URL Caching
Use STS (Strict Transport security)
Check for trusted certificates in key chain
Detect keyboard press caching
Use password strength meter
Use unpredictable session identifiers with high entropy
Enforce privacy policy
Use fast dormancy
Check anomalous usage pattern
Audit trail for access to paid resources
Record consent to PII data transfer

retrieved from the app, and a dynamic library for iOS and .smali code for Android is constructed for the specific app with these parameters. After the dylib / .smali module is created, it is copied to the app folder, and the required modifications are done to the app binary. The figure 7.10 shows the dynamic library getting loaded in an app-binary. When the loading process is complete, the app is recompiled/repackaged back to respective IPA or APK. The app is then signed with the individual certificates uploaded by the user. The new secure app is ready to be deployed to the Apple's app store or the Google's Play store.

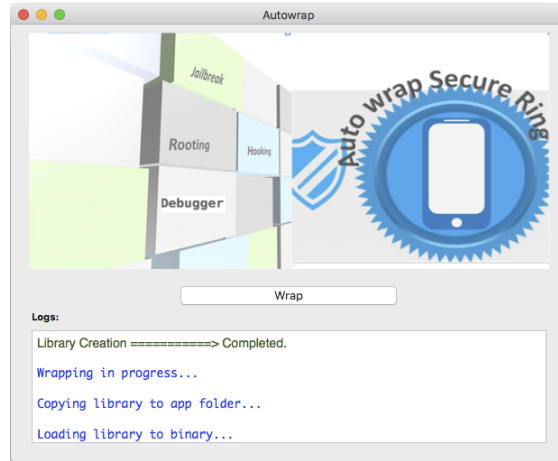


Figure 7.10: Library loaded into app binary

7.5 Results

7.5.1 Test results

Following are some of the android apps that were evaluated: ‘NBC News app’, ‘eRail transportation app’, and ‘Flipkart online shopping app’. The android apps were downloaded from the Play store. We analyzed these apps for various types of cyber-attacks and discovered several vulnerabilities. One of the most commonly found vulnerability was the lack of binary protection. These apps were running normally on rooted android devices. For the lack of binary protection, one of the solutions recommended by ‘MobiSecureWrap’ solution is to include the root detection solution. ‘MobiSecureWrap’ added the security mitigation functionality of root detection into the app, this solution alerts the user about the device being rooted and exits the app. Figure 7.11 shows NBC new apps after ‘MobiSecureWrap’ solution. Before securing the app, the ‘NBC news app’ was getting loaded without any checks on a rooted device.

Following are some of the iOS apps that were downloaded from the App Store: ‘BookMyShow’ (Movie Ticket booking app), ‘Times of India news app’ and ‘iCart’ (Online shopping app). Security assessments identified several vulnerabilities in these apps. In the case of iOS apps as well we found the lack of binary protection vulnerability, these apps were executing normally on a jailbroken iOS devices. The ‘MobiSecureWrap’ could not be executed directly on the apps since they were encrypted by Apple App store encryption. These apps were first decrypted on jailbroken iPhone using the tool called ‘Clutch’ [90]. After decrypting, ‘MobiSecureWrap’ tool incorporated the jailbreak detection solution into those apps successfully.

7.5.2 Performance test results

We evaluated the performance of the applications before and after wrapping the solution. In the case of Android devices, we installed the applications on a rooted device. We used ADB logcat

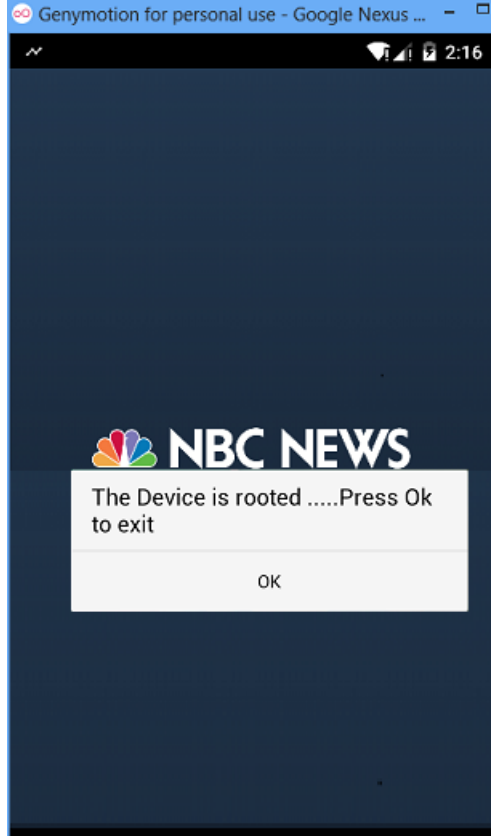


Figure 7.11: NBC with MobiSecureWrap

to check the time taken by applications to launch, on rooted devices and non-rooted devices. We recorded the time taken by each application to launch before including the solution. The applications were then wrapped with the solution for root detection and the time taken to launch the solution were measured again. Figure 7.12 shows that there is no significant difference in the launch time of the application with ‘MobiSecureWrap’ solution.

7.5.3 Machine learning results

To measure the model’s predictive ability, we used an approach to measure the accuracy by testing on a data set not used in the estimation. This data is called “Test Set” and the data used for the estimation is called “Training Set”. Mean Squared Error (MSE) [145] on the test set was used as the measure of predictive accuracy. If y_i is a vector of ‘i’ predictions, and \tilde{y}_i is of observed values corresponding to the inputs of the function which generated the predictions, then the MSE of the predictor can be estimated by:

$$MSE = \frac{1}{n} \sum_{i=1}^n (y_i - \tilde{y}_i)^2 \quad (7.2)$$

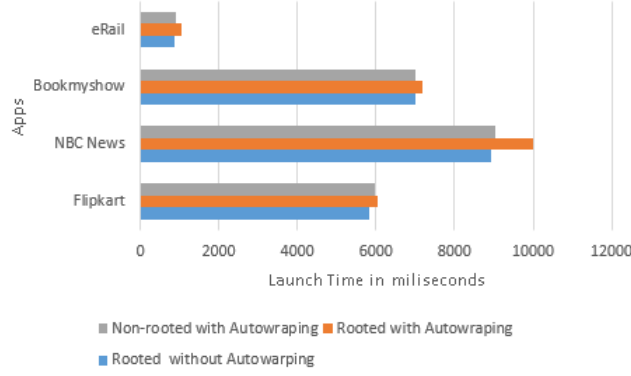


Figure 7.12: Performance test results

We employed k-fold cross-validation [78] for examining the accuracy of the model. The accuracy measures are accomplished as follows:

Assume there are n independent measurements, y_1, y_2, \dots, y_n . We considered $k = 5$.

1. First partition the dataset into k equal parts, say $(k_1, k_2, k_3, k_4, k_5)$.
2. Let observation set k_i form the test set and fit the model using the remaining data. Then compute the error e_i^* for the excluded observations.
3. Repeat step 1 and 2 for $i=1, \dots, k$.
4. Calculate the MSE from e_1^*, \dots, e_k^* . We shall call this the CV.

Table 7.5: Accuracy of solutions predicted

Solutions	Accuracy
Use encryption	94.9%
Use SQLcipher for SQL data Encryption	97.5%
Use certificates signed provided by a trusted CA provider	94.9%
Detect URL Caching	92.5%
Detect Keyboard press caching	95.0%
Local integrity checks to detect any unauthorized changes	95.0%
Avoid deprecated algorithm like RC4, MD5, MD4, and SHA1	97.5%
Average	95.3%

Here we split our training data into two sets: training data set and test data set. The training data set forms 80% of the total instances of the data, while test data forms 20% of the total instances of the data set. Here “ k ” specifies the number of times the cross-validation approach will be used. In each round, we took a different set of test data which is 20% of the total data set, such that, no instance of data is repeated for different rounds. This ensures that our algorithm

accuracy is tested on different kind of data. Considering the accuracy of the recommendation for all the solutions, taking the average of the accuracies, we obtained a mean accuracy of 95.3%. Refer to table 7.5. The authors obtained Precision of 0.875, Recall of 0.812 and F1 score of 0.84. The feedback system updates the training data to improve it further and thus, increases the accuracy of the recommendation.

7.6 Summary

In this chapter, we propose a new protocol to protect the existing iOS and Android mobile apps. In this method, the server will internally wrap the application in a secure envelope and translate the app into a new digital binary. The secure envelope mitigates the risks of cyber-attacks and at the same time reducing the total secure development lifecycle timeline. The wrapping process is carried out using well-defined protocols with no source code modifications in the original app. The original application features remain intact. This approach provides a quick and effective way of taking the apps to the market. Further, the solution eliminates the dependencies on the skilled resources for identifying and fixing the vulnerabilities. ‘MobileSecureWrap’ system can be used for an app developed for multiple form factors like mobile phones, tablet PC, and wearables. This system and method can be used to secure smartphone and wearable application binaries against cyberattack automatically. All of these contributions make our approach novel.

Chapter 8

Conclusion

In this chapter, we summarize our results presented in this thesis and suggest possible directions for future research.

8.1 Summary

The thesis demonstrates successful implementation of a practical '*Security Analysis and Digital Forensic Readiness System*' focused on the enterprise environment. The DFR and security was considered together to enhance the overall security of the mobile computing systems. The digital forensic readiness solution was implemented for Android and iOS. The solution focused on DFR at the operating system level, by securely preserving date and time stamps of targeted events running in smartphones, smartwatches, and wearables. These timestamps can be used to validate the digital evidence during a subsequent digital forensic investigation for any of the devices mentioned above.

For iOS, authors have shown how authentic copy of the MAC DTS values can be preserved on a local server or the cloud, which can help in offline digital forensics investigations in case of a security incident. The Dynamic library could detect malicious tampering of data by storing an authentic copy of the MAC DTS values of the artifacts on a secure location outside the iOS devices.

Further authors demonstrated that Android smartphones used in a BYOD environment are vulnerable to confidentiality and integrity attacks, where attackers can gain access to data stored on the phone and tamper it. The solution helps to detect this malicious tampering of data by storing an authentic copy of the MAC DTS values for selected files/ directories on a local server or the cloud and later verification / validation of the MAC DTS of questioned files/ directories. This solution can be easily applied to Android-based devices like smartwatches and wearables. The implemented solution uses kernel generated authentic timestamps stored at an outside se-

cured place for identifying malicious activities.

Further advancing DFR in the mobile computing devices, the author presented a novel form of forensic analysis technique, which analyzed over 5498 mobile ads to build the user profile that can be applied to identify the suspect who uses a particular device. The authors illustrated how Ads on a system could be used to identify a user. This solution collects ads information and creates a user behavior profile. Given the access to Mobile ads in any criminal investigation, forensic experts could use this information as one of the ways to construct the sequence of events and implicate the criminal. This is another system and method to authenticate or identify the user.

The security contribution of the thesis includes an automated security analysis solution for identifying potential mobile security threats. The proposed system is capable of identifying a mobile application (app) which is downloaded on an Android (phone, VM, or wearable) or an iOS (iPhone or iWatch) from an untrusted source. After identification, the system transfers the tainted app to a pre-deployed server for its security analysis which aims to identify top security threats that could have been exploited by hackers or malicious entities. The solution then automates the security analysis process, and suggested potential improvements to the existing app forensic analysis techniques. The authors have obtained a classification accuracy of 94.2 %, after applying machine learning techniques to predict vulnerabilities. The authors found that **‘M10-Lack of Binary Protection’** and **‘M3-Insufficient Transport Layer Protection’** are the most frequently encountered security threats because the application developers do not perform certificate inspection and fail to incorporate preventive measures against reverse engineering of their apps. Moreover, the apps belonging to **‘Retail’** and **‘Finance’** sectors were found to be highly vulnerable primarily because these apps have high exposure to customer’s PII information in addition to finance related information like bank account and card details. The situation gets critical with the increasing volume of customers who have started using these apps from the above-stated domains, in turn attracting more number attackers.

Further, a solution ‘SecureRing’ has been proposed for securing the mobile applications to provide an additional layer of protection against attacks. The solution utilizes the static library to detect the security vulnerabilities and takes appropriate actions. Thus with the proposed approach, any new or existing apps can be secured, even if the iOS device is jailbroken. With jail breaking on the rise and expected to continue, our work helps in addressing the security gap that exists. With enterprises adapting BYOD and jailbroken devices making its way into enterprises, our solution would help enterprises mitigate the security risks. The solution enables employees to use one device both for official and personal usage.

The author has also designed ‘MobiSecureWrap’, which is an automated solution for wrapping

mobile application binaries with additional security layer to protect them against potential threats. Authors propose a novel protocol wrapper which reduces the total SDLC time and protect the existing applications (immaterial of their features) while significantly decreasing the risks of cyber-attacks. The proposed rules resolve security threats in a way that typical dependencies on skilled resources can be removed. The server internally wraps the application in a secure envelope and then translates the application into a new digital binary while keeping its features intact. This shielding process is carried out seamlessly with no code changes in the original application, thus providing the application a faster time to market. The proposed system and method can handle applications of multiple form factors like mobile phones, tablet PC, and wearables. MobiSecureWrap recommends secure solutions based on detected security threats to protect the application binaries. The author evaluated over 5121 mobile applications to achieve a solution recommendation accuracy of 95.3%

8.2 Future work

In this section, we identify future research directions:

1. Identify tainted virtual instances of the corporate profile, initiate the efficient and incremental backup of the image, perform security and forensic analysis of the image on the cloud server.
2. Implement Anti Anti-forensics techniques to counter any Anti-forensic attacks.
3. Integration with a privacy preserving framework system.
4. Develop an *anomaly based intrusion detection system* for Android, by building models for expected behavior of applications on it. MAC DTS logs can be used as an important factor in building such models, so that the detection of malicious access can be accomplished in real time
5. Create a *customized kernel* using SE Linux and integrate as part of the MACDTS solution.
6. Ads analysis on different types of devices like desktop systems, laptop and tablet. Create a system and method to link the same user across different form factors.
7. Track and capture the live ads at the network layer, analyze these live ads to detect and prevent any crime before the act even occurs.
8. We further plan to develop the trust protocol implementation using Blockchain Distributed ledger. Using the distributed ledger would help the app developers, app owners and app users to verify the authenticity of the secure wrapper and to establish trust in the implementation.

Appendix A

Algorithm for logging MAC DTS using LKM

Algorithm 1 Algorithm for logging MAC DTS using LKM

```
Function root_start
Output: file pointer to record.log file
{ Log File Pointer = File Open ("/data/data/record.log")
  Original Sys Open Pointer = sys_call_table[__NR_open]
  Original Sys Unlink Pointer = sys_call_table[__NR_unlink]
  sys_call_table[__NR_open] = (unsigned long) Our Sys Open call hacked_open
  sys_call_table[__NR_unlink] = (unsigned long) Our Sys Open call hacked_unlink }

Function hacked_open
Input: file Path Name, Flags
Output: file Pointer to record.log file
{ Filter = File/ Folder path for enabling time stamp logging
  Path Name = Current working directory
if Filter = Path Name then
  Log Path Name, Date and Time and File Operation Type ID
else
  Return to Original Sys Open Operation
}

Function hacked_unlink
Input: file Path Name
{
if Path Name = Log File Path then
  Return -1 ... /* Deny log file deletion. */
else
  Return to Original Sys Unlink Operation
}

Function root_stop
Original Sys Open Pointer, Log File Pointer
{ Restore to Original System Call
  sys_call_table[__NR_open] = Original Sys Open Pointer
  sys_call_table[__NR_unlink] = Original Sys Unlink Pointer
  Close Log File Pointer }
```

Bibliography

- [1] ISO/IEC JTC 1/SC 27. Iso/iec 27043:2015(en), information technology — security techniques — incident investigation principles and processes. <https://www.iso.org/obp/ui/#iso:std:iso-iec:27043:ed-1:v1:en>, March 2015. Accessed: September 09, 2018.
- [2] Nurul Hidayah Ab Rahman, William Bradley Glisson, Yanjiang Yang, and Kim-Kwang Raymond Choo. Forensic-by-design framework for cyber-physical cloud systems. *IEEE Cloud Computing*, 3(1):50–59, 2016.
- [3] Dmitrijs Abalenkovs, Petro Bondarenko, Vijay Kumarraju Pathapati, André Nordbø, Dmytro Piatkivskyi, John Erik Rekdal, and Pieter Bloemerus Ruthven. Mobile forensics: Comparison of extraction and analyzing methods of ios and android. *Gjovik University College, Gjovik, Norway*, 2012.
- [4] Diquet Alban, Chow Angela, and Castro Eric. Code injection on ios 8 for the greater good. In *BlackHat 2015*, 2015.
- [5] Pietro Albano, Aniello Castiglione, Giuseppe Cattaneo, and Alfredo De Santis. A novel anti-forensics technique for the android os. In *Broadband and wireless computing, communication and applications (bwcca), 2011 international conference on*, pages 380–385. IEEE, 2011.
- [6] Ahmed Alenezi, Nurul Huda Nik Zulkipli, Hany F Atlam, Robert John Walters, and Gary B Wills. The impact of cloud forensic readiness on security. In *CLOSER*, pages 511–517, 2017.
- [7] Soltan Abed Alharbi. *Proactive System for Digital Forensic Investigation*. PhD thesis, University of Victoria, 2014.
- [8] Sahel Alounch, Heba Bsoul, and Mazen Kharbutli. Protecting binary files from stack-based buffer overflow. In *Information Science and Applications*, pages 415–422. Springer, 2015.
- [9] Md Sarfaraj Alam Ansari, Arijit Chattopadhyay, and Suvrojit Das. A kernel level vfs logger for building efficient file system intrusion detection system. In *Computer and Network Technology (ICCNT), 2010 Second International Conference on*, pages 273–279. IEEE, 2010.

- [10] Appdome. Secure mobile apps without the work. <https://www.appdome.com/solutions/security-integrations>, June 2018. Accessed: June 09, 2018.
- [11] Shiva Azadegan, Wei Yu, Hui Liu, M Sistani, and Subrata Acharya. Novel anti-forensics approaches for smart phones. In *System Science (HICSS), 2012 45th Hawaii International Conference on*, pages 5424–5431. IEEE, 2012.
- [12] Satish B. Penetration testing for iphone applications – part 5, infosec institute, inc., elmwood park, illinois, usa. <http://resources.infosecinstitute.com/penetration-testing-for-iphone-applications-part-5/>, 2013. Accessed: June 09, 2018.
- [13] Paul Barford, Igor Canadi, Darja Krushevskaja, Qiang Ma, and S Muthukrishnan. Adscape: Harvesting and analyzing online display ads. In *Proceedings of the 23rd international conference on World wide web*, pages 597–608. ACM, 2014.
- [14] Mridul Sankar Barik, Gaurav Gupta, Shubhro Sinha, Alok Mishra, and Chandan Mazumdar. An efficient technique for enhancing forensic capabilities of ext2 file system. *digital investigation*, 4:55–61, 2007.
- [15] Venture Beat. Apps downloaded from the google play in 2017. <https://venturebeat.com/2018/05/08/with-94-billion-installs-in-2017-google-helps-android-developers-shrink-their-apps/>, May 2018. Accessed: June 09, 2018.
- [16] Michael Becher, Felix C Freiling, Johannes Hoffmann, Thorsten Holz, Sebastian Uellenbeck, and Christopher Wolf. Mobile security catching up? revealing the nuts and bolts of the security of mobile devices. In *Security and Privacy (SP), 2011 IEEE Symposium on*, pages 96–111. IEEE, 2011.
- [17] Jeff Benjamin. Jailbreak ios 7.1.x with pangu 1.1 on windows. <http://www.idownloadblog.com/2014/06/29/how-to-jailbreak-ios-7-1-x-with-pangu-1-1-on-windows-video/>, June 2014. Accessed: June 09, 2018.
- [18] Theodore Book and Dan S Wallach. An empirical study of mobile ad targeting. *arXiv preprint arXiv:1502.06577*, 2015.
- [19] brandontreb. Beginning jailbroken ios development - building and deployment. <http://brandontreb.com/beginning-jailbroken-ios-development-building-and-deployment>, April 2011. Accessed on: June 09, 2018.
- [20] Florian P Buchholz and Courtney Falk. Design and implementation of zeitline: a forensic timeline editor. In *DFRWS*, 2005.

- [21] Brian Carrier and Eugene H Spafford. An event-based digital forensic investigation framework. In *Digital forensic research workshop*, pages 11–13, 2004.
- [22] Claude Castelluccia, Mohamed-Ali Kaafar, and Minh-Dung Tran. Betrayed by your ads! In *International Symposium on Privacy Enhancing Technologies*, pages 1–17. Springer, 2012.
- [23] Edwin Chen. Choosing a machine learning classifier. <http://blog.echen.me/2011/04/27/choosing-a-machine-learning-classifier/>, January 2011. Accessed: August 09, 2018.
- [24] Erika Chin, Adrienne Porter Felt, Kate Greenwood, and David Wagner. Analyzing inter-application communication in android. In *Proceedings of the 9th international conference on Mobile systems, applications, and services*, pages 239–252. ACM, 2011.
- [25] Aniello Cimitile, Fabio Martinelli, and Francesco Mercaldo. Machine learning meets ios malware: Identifying malicious applications on apple environment. In *ICISSP*, pages 487–492, 2017.
- [26] Android community. Loadable kernel modules. <https://source.android.com/devices/architecture/kernel/modular-kernels#loadable-kernel-modules>, August 2018. Accessed on: August 09, 2018.
- [27] HTC Corporation. The htc developer center, taoyuan city, taiwan. www.htcdev.com/devcenter/downloads, May 2013. Accessed: June 09, 2018.
- [28] Samsung Electronics Corporation. Samsung open source release center, suwon, south korea. opensource.samsung.com/, January 2013. Accessed: June 09, 2018.
- [29] Suvrojit Das, Arijit Chattopadhyay, Dipesh Kumar Kalyani, and Monojit Saha. File-system intrusion detection by preserving mac dts: a loadable kernel module based approach for linux kernel 2.6. x. In *Proceedings of the 5th Annual Workshop on Cyber Security and Information Intelligence Research: Cyber Security and Information Intelligence Challenges and Strategies*, page 57. ACM, 2009.
- [30] Lucia De Marco, Filomena Ferrucci, and Tahar Kechadi. Reference architecture for a cloud forensic readiness system. *ICST*, 2014.
- [31] Checkpoint Dimensional Research. The impact of mobile devices on information security: A survey of it and security professionals. october 2014. <https://www.checkpoint.com/downloads/product-related/report/check-point-capsule-2014-mobile-security-survey-report.pdf>, October 2014. Accessed: June 09, 2018.
- [32] Alban Diquet. Trustkit, ios 9 and the shared cache. <https://nabla-c0d3.github.io/blog/2015/10/17/trustkit-ios-9-shared-cache/>, October 2015. Accessed: August 09, 2018.

- [33] Alessandro Distefano, Gianluigi Me, and Francesco Pace. Android anti-forensics through a local paradigm. *digital investigation*, 7:S83–S94, 2010.
- [34] Julian T Dolby, Pietro Ferrara, Marco Pistoia, and Omer Tripp. Install-time security analysis of mobile applications, February 8 2018. US Patent App. 15/231,093.
- [35] Christian D’Orazio, Aswami Ariffin, and Kim-Kwang Raymond Choo. ios anti-forensics: How can we securely conceal, delete and insert data? In *System Sciences (HICSS), 2014 47th Hawaii International Conference on*, pages 4838–4847. IEEE, 2014.
- [36] Josiah Dykstra and Alan T Sherman. Acquiring forensic evidence from infrastructure-as-a-service cloud computing: Exploring and evaluating tools, trust, and techniques. *Digital Investigation*, 9:S90–S98, 2012.
- [37] Josiah Dykstra and Alan T Sherman. Design and implementation of frost: Digital forensic tools for the openstack cloud computing platform. *Digital Investigation*, 10:S87–S95, 2013.
- [38] Mohamed Elyas, Atif Ahmad, Sean B Maynard, and Andrew Lonie. Digital forensic readiness: Expert perspectives on a theoretical framework. *Computers & Security*, 52:70–89, 2015.
- [39] William Enck, Damien Ocateau, Patrick McDaniel, and Swarat Chaudhuri. A study of android application security. In *USENIX security symposium*, volume 2, page 2, 2011.
- [40] Barbara Endicott-Popovsky, Deborah A Frincke, and Carol A Taylor. A theoretical framework for organizational network forensic readiness. *Journal of Computers*, 2(3):1–11, 2007.
- [41] Stefan Esser. Exploiting the ios kernel. In *Proceedings of the Black Hat Technical Security Conference: USA*, 2011.
- [42] Facebook. Fishhook. <https://github.com/facebook/fishhook>, April 2018. Accessed: August 09, 2018.
- [43] Adrienne Porter Felt, Erika Chin, Steve Hanna, Dawn Song, and David Wagner. Android permissions demystified. In *Proceedings of the 18th ACM conference on Computer and communications security*, pages 627–638. ACM, 2011.
- [44] Adam P Fuchs, Avik Chaudhuri, and Jeffrey S Foster. Scandroid: Automated security certification of android. Technical report, University of Maryland, 2009.
- [45] Eran Gal and Sivan Toledo. Algorithms and data structures for flash memories. *ACM Computing Surveys (CSUR)*, 37(2):138–163, 2005.
- [46] Gartner. Predicts by 2017, half of employers would adopt byod. <https://www.gartner.com/newsroom/id/2466615>, May 2013. Accessed: June 09, 2018.
- [47] Gartner. 75 percent of mobile applications will fail basic security tests. <https://www.gartner.com/newsroom/id/2846017>, September 2014. Accessed: August 09, 2018.

- [48] Gartner. 8.4 billion connected "things" will be in use in 2017. <https://www.gartner.com/newsroom/id/3598917>, February 2017. Accessed: June 09, 2018.
- [49] Dimitris Geneiatakis, Igor Nai Fovino, Ioannis Kounelis, and Paquale Stirparo. A permission verification approach for android mobile applications. *Computers & Security*, 49:192–205, 2015.
- [50] Jayaprakash Govindaraj, Rashmi Mata, Robin Verma, and Gaurav Gupta. Forensic-ready secure ios apps for jailbroken iphones. In *IFIP International Conference on Digital Forensics*, pages 235–249. Springer, 2015.
- [51] Jayaprakash Govindaraj, Robin Verma, and Gaurav Gupta. Analyzing mobile device ads to identify users. In *IFIP International Conference on Digital Forensics*, pages 107–126. Springer, 2016.
- [52] Jayaprakash Govindaraj, Robin Verma, and Gaurav Gupta. Precognition: Automated digital forensic readiness system for mobile computing devices in enterprises. In *Annual ADFSL Conference on Digital Forensics, Security and Law. 11*, 2018.
- [53] Jayaprakash Govindaraj, Robin Verma, R Mata, and Gaurav Gupta. Poster: isecurering: Forensic ready secure ios apps for jailbroken iphones. In *35th IEEE Symposium on Security and Privacy*, 2014.
- [54] Tim Grance, Karen Kent, and Brian Kim. Computer security incident handling guide. *NIST Special Publication*, 800:61, 2004.
- [55] CP Grobler and CP Louwrens. Digital forensic readiness as a component of information security best practice. In *New Approaches for Security, Privacy and Trust in Complex Environments*, pages 13–24. Springer, 2007.
- [56] Miniwatts Marketing Group. Internet usage statistics 2017. <http://www.internetworldstats.com/stats.htm>, December 2017. Accessed: June 09, 2018.
- [57] Justin Grover. Android forensics: Automated data collection and reporting from a mobile device. *Digital Investigation*, 10:S12–S20, 2013.
- [58] Alessandro Guarino. Digital forensics as a big data challenge. In *ISSE 2013 securing electronic business processes*, pages 197–203. Springer, 2013.
- [59] Mark Guido, Jared Ondricek, Justin Grover, David Wilburn, Thanh Nguyen, and Andrew Hunt. Automated identification of installed malicious android applications. *Digital Investigation*, 10:S96–S104, 2013.
- [60] Anshul Gupta, Carolina Milanese, Roberta Cozza, and CK Lu. Market share analysis: Mobile phones, worldwide. *2Q13, Gartner Report*, 2013.

- [61] Omar Hamoui. Targeting an ad to a mobile device, March 6 2008. US Patent App. 11/702,958.
- [62] Ryan Harris. Arriving at an anti-forensics consensus: Examining how to define and control the anti-forensics problem. *digital investigation*, 3:44–49, 2006.
- [63] Basel Hasan, Tariq Mahmoud, Jorge Marx Gómez, Reshmi Pramod, and Joachim Kurzhöfer. User acceptance identification of restrictions caused by mobile security countermeasures. *MOBILITY 2015*, page 39, 2015.
- [64] Bryan Henderson. Linux loadable kernel module howto. *The Linux Docu*, 2001.
- [65] Michael Howard and Steve Lipner. *The security development lifecycle*, volume 8. Microsoft Press Redmond, 2006.
- [66] Dustin Howett. Debugserver - console app that acts as server for remote gdb or lldb debugging. <http://iphonedevwiki.net/index.php/Debugserver>. Accessed: June 09, 2018.
- [67] Dustin Howett. Theos - cross-platform suite of development tools for managing, developing, and deploying ios software without the use of xcode. <http://iphonedevwiki.net/index.php/Theos>. Accessed: June 09, 2018.
- [68] Infosec Institute Inc. Ios application security part 2 – getting class information of ios apps, elmwood park, illinois, usa. <http://resources.infosecinstitute.com/ios-application-security-part-2-getting-class-information-of-ios-apps/>. Accessed: June 09, 2018.
- [69] GSMA Intelligence. Number of mobile subscribers worldwide hits 5 billion. <https://www.gsma.com/newsroom/press-release/number-mobile-subscribers-worldwide-hits-5-billion/>, June 2017. Accessed: June 09, 2018.
- [70] Apple iOS Edition. The art of building bulletproof mobile apps. *CiteSeerX*, 2012.
- [71] Saurik iPhone Wiki. Xcon - solution for hooking every known method and function responsible for informing an application of a jailbroken device. <https://www.theiphonewiki.com/wiki/XCon>. Accessed: June 09, 2018.
- [72] iPhoneHacks. Jailbreaking your iphone remains legal in us. <http://www.iphonehacks.com/2012/10/jailbreaking-iphone-legal-in-us-illegal-to-jailbreak-ipad-unlock-iphones.html>, October 2012. Accessed: June 09, 2018.
- [73] Anurag Kumar Jain and Devendra Shanbhag. Addressing security and privacy risks in mobile applications. *IT Professional*, 14(5):28–33, 2012.

- [74] Victor R Kebande and Hein S Venter. Novel digital forensic readiness technique in the cloud environment. *Australian Journal of Forensic Sciences*, 50(5):552–591, 2018.
- [75] Victor Rigworo Kebande, Nickson Karie Menza, and Hein S Venter. Functional requirements for adding digital forensic readiness as a security component in iot environments. *International Journal on Advanced Science, Engineering and Information Technology*, 8(2):342–349, 2018.
- [76] James King. Android application security with owasp mobile top 10 2014, 2014.
- [77] Ron Kohavi. Scaling up the accuracy of naive-bayes classifiers: a decision-tree hybrid. In *KDD*, volume 96, pages 202–207. Citeseer, 1996.
- [78] Ron Kohavi et al. A study of cross-validation and bootstrap for accuracy estimation and model selection. *Ijcai*, 14(2):1137–1145, 1995.
- [79] Aleksandra Korolova. Privacy violations using microtargeted ads: A case study. In *Data Mining Workshops (ICDMW), 2010 IEEE International Conference on*, pages 474–482. IEEE, 2010.
- [80] Mariantonietta La Polla, Fabio Martinelli, and Daniele Sgandurra. A survey on security for mobile devices. *Communications Surveys & Tutorials, IEEE*, 15(1):446–471, 2013.
- [81] Leandro Marinho Lars Shmidt Thieme. Multi-label classification. <https://www.ismll.uni-hildesheim.de/lehre/ml-06w/skript/ml-4up-04-mlabelclassification.pdf>, January 2007. Accessed on: June 09, 2018.
- [82] Joe Laszlo, IAB Mobile Advertising Committee, et al. The new unwired world: An iab status report on mobile advertising. *Journal of Advertising Research*, 49(1):27–43, 2009.
- [83] Li Li, Alexandre Bartel, Jacques Klein, and Yves Le Traon. Detecting privacy leaks in android apps, 2014.
- [84] Li Li, Tegawendé F Bissyandé, Mike Papadakis, Siegfried Rasthofer, Alexandre Bartel, Damien Octeau, Jacques Klein, and Le Traon. Static analysis of android apps: A systematic literature review. *Information and Software Technology*, 88:67–95, 2017.
- [85] Michael Mainelli and Mike Smith. Sharing ledgers for sharing economies: an exploration of mutual distributed ledgers (aka blockchain technology). *The Journal of Financial Perspectives: FinTech*, 2015.
- [86] Simone Margaritelli. Android native api hooking with library injection and elf introspection. <https://www.evilsocket.net/2015/05/04/android-native-api-hooking-with-library-injecto/>, May 2014. Accessed: June 09, 2018.

- [87] Andrew Marrington, Ibrahim Baggili, George Mohay, and Andrew Clark. Cat detect (computer activity timeline detection): A tool for detecting inconsistency in computer activity timelines. *digital investigation*, 8:S52–S61, 2011.
- [88] Mattt. ‘trustkit, ios 9 and the shared cache. <https://nshipster.com/method-swizzling/>, February 2014. Accessed: August 09, 2018.
- [89] Erika McCallister. *Guide to protecting the confidentiality of personally identifiable information*. Diane Publishing, 2010.
- [90] Michail Menesidis and Μιχαήλ Μενεσιδης. ios forensics & data leakage. Master’s thesis, Πανεπιστήμιο Πειραιώς, 2016.
- [91] Charlie Miller. Owing the fanboys: Hacking mac os x,. www.blackhat.com/presentations/bh-jp-08/bh-jp-08-Miller/BlackHat-Japan-08-Miller-Hacking-OSX.pdf, January 2008. Accessed: June 09, 2018.
- [92] Charlie Miller. Mobile attacks and defense. *Security & Privacy, IEEE*, 9(4):68–70, 2011.
- [93] Keith W Miller, Jeffrey Voas, and George F Hurlburt. Byod: Security and privacy considerations. *It Professional*, 14(5):53–55, 2012.
- [94] George Mohay. Technical challenges and directions for digital forensics. In *Systematic Approaches to Digital Forensic Engineering, 2005. First International Workshop on*, pages 155–161. IEEE, 2005.
- [95] Francois Mouton and HS Venter. A prototype for achieving digital forensic readiness on wireless sensor networks. In *AFRICON, 2011*, pages 1–6. IEEE, 2011.
- [96] Alexios Mylonas, Vasilis Meletiadis, Bill Tsoumas, Lilian Mitrou, and Dimitris Gritzalis. Smartphone forensics: A proactive investigation scheme for evidence acquisition. In *Information Security and Privacy Research*, pages 249–260. Springer, 2012.
- [97] K. Nagamine. Idc worldwide mobile phone tracker. *International Data Corporation, Framingham, Massachusetts, USA, 2013*, 2013.
- [98] Suman Nath. Madscope: Characterizing mobile in-app targeted ads. In *Proceedings of the 13th Annual International Conference on Mobile Systems, Applications, and Services*, pages 59–73. ACM, 2015.
- [99] Steve Nygard. Class-dump. <http://stevenygard.com/projects/class-dump/>, November 2013. Accessed: Sep 09, 2018.
- [100] Earl Oliver. A survey of platforms for mobile networks research. *SIGMOBILE Mob. Comput. Commun. Rev.*, 12(4):56–63, February 2009.

- [101] Jens Olsson and Martin Boldt. Computer forensic timeline visualization tool. *digital investigation*, 6:S78–S87, 2009.
- [102] Myat Nandar Oo and Thandar Thein. Forensic readiness on hadoop platform: Non-ambari hdp as a case study. *IJCSIS*, 2017.
- [103] OWASP. Owasp mobile top 10 2016-top 10. https://www.owasp.org/index.php/Mobile_Top_10_2016-Top_10, February 2017. Accessed: August 09, 2018.
- [104] OWASP. Owasp top 10 mobile controls and design principles. https://www.owasp.org/index.php/OWASP_Mobile_Security_Project#tab=Top_10_Mobile_Controls, April 2017. Accessed: August 09, 2018.
- [105] OWASP. Owasp enterprise security api. https://www.owasp.org/index.php/Category:OWASP_Enterprise_Security_API, August 2018. Accessed: August 09, 2018.
- [106] Pádraig O’Sullivan, Kapil Anand, Aparna Kotha, Matthew Smithson, Rajeev Barua, and Angelos D Keromytis. Retrofitting security in cots software with binary rewriting. In *IFIP International Information Security Conference*, pages 154–172. Springer, 2011.
- [107] George Pangalos, Christos Ilioudis, and Ioannis Pagkalos. The importance of corporate forensic readiness in the information security framework. In *Enabling Technologies: Infrastructures for Collaborative Enterprises (WETICE), 2010 19th IEEE International Workshop on*, pages 12–16. IEEE, 2010.
- [108] Christian Papathanasiou and Nicholas J Percoco. This is not the droid you’re looking for. *Def Con*, 18, 2010.
- [109] Sungmi Park, Nikolay Akatyev, Yunsik Jang, Jisoo Hwang, Donghyun Kim, Woonseon Yu, Hyunwoo Shin, Changhee Han, and Jonghyun Kim. A comparative study on data protection legislations and government standards to implement digital forensic readiness as mandatory requirement. *Digital Investigation*, 24:S93–S100, 2018.
- [110] Liliana Pasquale, Dalal Alrajeh, Claudia Peersman, Thein Tun, Bashar Nuseibeh, and Awais Rashid. Towards forensic-ready software systems. In *Proceedings of the 40th International Conference on Software Engineering: New Ideas and Emerging Results*, pages 9–12. ACM, 2018.
- [111] Liliana Pasquale, Carlo Ghezzi, Claudio Menghi, Christos Tsigkanos, and Bashar Nuseibeh. Topology aware adaptive security. In *Proceedings of the 9th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, pages 43–48. ACM, 2014.
- [112] Marco Pistoia, Omer Tripp, and David Lubensky. Combining static code analysis and machine learning for automatic detection of security vulnerabilities in mobile apps. In *Application Development and Design: Concepts, Methodologies, Tools, and Applications*, pages 1121–1147. IGI Global, 2018.

- [113] Joel Reardon, Srdjan Capkun, and David Basin. Data node encrypted file system: Efficient secure deletion for flash memory. In *Proceedings of the 21st USENIX conference on Security symposium*, pages 17–17. USENIX Association, 2012.
- [114] Kamil Reddy and Hein S Venter. The architecture of a digital forensic readiness management system. *Computers & Security*, 32:73–89, 2013.
- [115] Mathieu RENARD. Practical ios apps hacking. *G 2 reHack 012*, page 14, 2012.
- [116] Robert Rowlingson. A ten step process for forensic readiness. *International Journal of Digital Evidence*, 2(3):1–28, 2004.
- [117] Keyun Ruan, Joe Carthy, Tahar Kechadi, and Ibrahim Baggili. Cloud forensics definitions and critical criteria for cloud forensic capability: An overview of survey results. *Digital Investigation*, 10(1):34–43, 2013.
- [118] Sarker T Ahmed Rumeen and Donggang Liu. Droidtest: Testing android applications for leakage of private information. In *Information Security*, pages 341–353. Springer, 2015.
- [119] Jason Sachowski. *Implementing Digital Forensic Readiness: From Reactive to Proactive Process*. Syngress, 2016.
- [120] Alireza Sadeghi, Hamid Bagheri, Joshua Garcia, and Sam Malek. A taxonomy and qualitative comparison of program analysis techniques for security assessment of android software. *IEEE Transactions on Software Engineering*, 43(6):492–530, 2017.
- [121] SaurikIT. Cydia substrate. <http://www.cydiasubstrate.com>. Accessed: June 09, 2018.
- [122] LLC SaurikIT. Cydia substrate, the powerful code modification platform behind cydia. http://iphonedevwiki.net/index.php/Cydia_Substrate, 2016.
- [123] Sebastián Guerrero Selma. Hacking ios on the run: Using cycrypt,. In *RSA Conference 2014*, 2014.
- [124] Asaf Shabtai, Yuval Fledel, and Yuval Elovici. Automated static code analysis for classifying android applications using machine learning. In *Computational Intelligence and Security (CIS), 2010 International Conference on*, pages 329–333. IEEE, 2010.
- [125] Asaf Shabtai, Yuval Fledel, and Yuval Elovici. Securing android-powered mobile devices using selinux. *IEEE Security & Privacy*, 8(3):36–44, 2010.
- [126] Stephen Smalley, Timothy Fraser, and Chris Vance. Linux security modules: General security hooks for linux, 2001.
- [127] Stephen Smalley and Trust Mechanisms R2X. The case for se android. *Linux Security Summit*, 2011.
- [128] Aaron Smith. Pew research center. *Smartphone Ownership–2013 Update*, 5, 2013.

- [129] Aaron Smith. Smartphone ownership–2013 update. *Pew Research Center: Washington DC*, 12, 2013.
- [130] Statista. Apps downloaded from the apple app store as of june 2017. <https://www.statista.com/statistics/263794/number-of-downloads-from-the-apple-app-store/>, June 2017. Accessed: June 09, 2018.
- [131] Statista. Number of social media users worldwide from 2010 to 2021 (in billions). <https://www.statista.com/statistics/278414/number-of-worldwide-social-network-users/>, June 2017. Accessed: June 09, 2018.
- [132] San-Tsai Sun, Andrea Cuadros, and Konstantin Beznosov. Android rooting: Methods, detection, and evasion. In *Proceedings of the 5th Annual ACM CCS Workshop on Security and Privacy in Smartphones and Mobile Devices*, pages 3–14. ACM, 2015.
- [133] Alfred Aho Suzanna Schmeelk. Static analysis techniques used in android application security analysis. http://www.cs.columbia.edu/~aho/cs6998/Lectures/14-12-01_Schmeelk_AndroidSecurity.pptx, December 2014. Accessed on: June 09, 2018.
- [134] John Tan. Forensic readiness. *Cambridge, MA:@ Stake*, pages 1–23, 2001.
- [135] Theodora H Titonis, Nelson R Manohar-Alers, and Christopher J Wysopal. Automated behavioral and static analysis using an instrumented sandbox and machine learning classification for mobile security, January 25 2018. US Patent App. 15/596,461.
- [136] Vincent Toubiana, Vincent Verdot, and Benoit Christophe. Cookie-based privacy issues on google services. In *Proceedings of the second ACM conference on Data and Application Security and Privacy*, pages 141–148. ACM, 2012.
- [137] Philip M Trenwith and Hein S Venter. Digital forensic readiness in the cloud. In *Information Security for South Africa, 2013*, pages 1–5. IEEE, 2013.
- [138] Aleksandar Valjarevic and Hein S Venter. Towards a digital forensic readiness framework for public key infrastructure systems. In *Information Security South Africa (ISSA), 2011*, pages 1–10. IEEE, 2011.
- [139] Robin Verma, Jayaprakash Govindaraj, and Gaurav Gupta. Preserving dates and timestamps for incident handling in android smartphones. In *Advances in Digital Forensics X*, pages 209–225. Springer, 2014.
- [140] Wei Wang, Yuanyuan Li, Xing Wang, Jiqiang Liu, and Xiangliang Zhang. Detecting android malicious apps and categorizing benign apps with ensemble of classifiers. *Future Generation Computer Systems*, 78:987–994, 2018.

- [141] Patrick Wardle. Dylib hijacking on os x. In *CanSecWest 2015*, 2015.
- [142] Fengguo Wei, Sankardas Roy, Xinming Ou, et al. Amandroid: A precise and general inter-component data flow analysis framework for security vetting of android apps. *ACM Transactions on Privacy and Security (TOPS)*, 21(3):14, 2018.
- [143] Michael C Weil. Dynamic time & date stamp analysis. *International Journal of Digital Evidence*, 1(2), 2002.
- [144] David A Willis. Bring your own device: the facts and the future. *Gartner Inc*, 2013.
- [145] Cort J Willmott and Kenji Matsuura. Advantages of the mean absolute error (mae) over the root mean square error (rmse) in assessing average model performance. *Climate research*, 30(1):79–82, 2005.
- [146] R Winsniewski. Android-apktool: A tool for reverse engineering android apk files, 2012.
- [147] Zhi Xu, Xinran Wang, and Huagang Xie. Static and dynamic security analysis of apps for mobile devices, October 18 2018. US Patent App. 16/005,441.
- [148] Divakar Yadav, Mahesh K Mishra, and Sachin Prakash. Mobile forensics challenges and admissibility of electronic evidences in india. In *Computational Intelligence and Communication Networks (CICN), 2013 5th International Conference on*, pages 237–242. IEEE, 2013.
- [149] Jun Yan, Ning Liu, Gang Wang, Wen Zhang, Yun Jiang, and Zheng Chen. How much can behavioral targeting help online advertising? In *Proceedings of the 18th international conference on World wide web*, pages 261–270. ACM, 2009.
- [150] Jay Hyunjae Yu. You’ve got mobile ads! young consumers’responses to mobile ads with different types of interactivity. *International Journal of Mobile Marketing*, 8(1), 2013.
- [151] Jonathan Zdziarski. *Hacking and securing iOS applications: stealing data, hijacking software, and how to prevent it*. " O’Reilly Media, Inc.", 2012.