Understanding the Behaviour of Privacy in Mobile Apps and Detecting Privacy Leaks

Dissertation-II submitted in fulfilment of the requirements for the Degree of

MASTER OF TECHNOLOGY

in

COMPUTER SCIENCE AND ENGINEERING

By

SUMIT KUMAR

11600788

Supervisor

Mr. RAVISHANKER

School of Computer Science and Engineering

Lovely Professional University

Phagwara, Punjab (India)

Month November-December Year 2017

LOVELY PROFESSIONAL UNIVERSITY
Transforming Education, Transforming India

TOPIC APPROVAL PERFORMA

School of Computer Science and Engineering

Program : P172::M.Tech. (Computer Science and Engineering) [Full Time]

COURSE CODE : CSE548 **REGULAR/BACKLOG :** Regular **GROUP NUMBER :** CSERGD0056

Supervisor Name : Ravishanker **UID :** 12412 **Designation :** Assistant Professor

Qualification : M.Tech (IT) **Research Experience :** 8 years

SR.NO.	NAME OF STUDENT	REGISTRATION NO	BATCH	SECTION	CONTACT NUMBER
1	Sumit Kumar	11600788	2016	K1637	8130330964

SPECIALIZATION AREA : Networking and Security **Supervisor Signature:** Ravishanker 12412

PROPOSED TOPIC : Understanding the behavior of privacy in mobile apps and detecting privacy leaks

Qualitative Assessment of Proposed Topic by PAC		
Sr.No.	Parameter	Rating (out of 10)
1	Project Novelty: Potential of the project to create new knowledge	7.00
2	Project Feasibility: Project can be timely carried out in-house with low-cost and available resources in the University by the students.	7.50
3	Project Academic Inputs: Project topic is relevant and makes extensive use of academic inputs in UG program and serves as a culminating effort for core study area of the degree program.	7.00
4	Project Supervision: Project supervisor's is technically competent to guide students, resolve any issues, and impart necessary skills.	7.25
5	Social Applicability: Project work intends to solve a practical problem.	7.25
6	Future Scope: Project has potential to become basis of future research work, publication or patent.	7.75

PAC Committee Members		
PAC Member 1 Name: Prateek Agrawal	UID: 13714	Recommended (Y/N): NA
PAC Member 2 Name: Pushpendra Kumar Pateriya	UID: 14623	Recommended (Y/N): Yes
PAC Member 3 Name: Deepak Prashar	UID: 13897	Recommended (Y/N): Yes
PAC Member 4 Name: Kewal Krishan	UID: 11179	Recommended (Y/N): Yes
PAC Member 5 Name: Anupinder Singh	UID: 19385	Recommended (Y/N): NA
DAA Nominee Name: Kanwar Preet Singh	UID: 15367	Recommended (Y/N): Yes

Final Topic Approved by PAC: **Understanding the behavior of privacy in mobile apps and detecting privacy leaks**

Overall Remarks: Approved

PAC CHAIRPERSON Name: 11024::Amandeep Nagpal **Approval Date:** 05 Mar 2017

11/24/2017 3:10:46 PM

DECLARATION STATEMENT

I hereby declare that the research work reported in the dissertation-II entitled "UNDERSTANDING THE BEHAVIOUR OF PRIVACY IN MOBILE APPS AND DETECTING PRIVACY LEAKS" in partial fulfilment of the requirement for the award of Degree for Master of Technology in Computer Science and Engineering at Lovely Professional University, Phagwara, Punjab is an authentic work carried out under supervision of my research supervisor Mr. RaviShanker. I have not submitted this work elsewhere for any degree or diploma.

I understand that the work presented herewith is in direct compliance with Lovely Professional University's Policy on plagiarism, intellectual property rights, and highest standards of moral and ethical conduct. Therefore, to the best of my knowledge, the content of this dissertation represents authentic and honest research effort conducted, in its entirety, by me. I am fully responsible for the contents of my dissertation work.

Signature of Candidate

Sumit Kumar

11600788

SUPERVISOR'S CERTIFICATE

This is to certify that the work reported in the M.Tech Dissertation-II entitled "**UNDERSTANDING THE BEHAVIOUR OF PRIVACY IN MOBILE APPS AND DETECTING PRIVACY LEAKS"**, submitted by **Sumit Kumar** at **Lovely Professional University, Phagwara, India** is a bonafide record of his original work carried out under my supervision. This work has not been submitted elsewhere for any other degree.

Signature of Supervisor

RaviShanker

Date:

ACKNOWLEDGMENTS

The report has been written with the kind guidance and support of my mentor. The satisfaction and happiness that accompany the successful completion of any task would be incomplete without the names of people who made it possible, whose constant guidance and encouragement crowns all efforts with our success.

I would like to present my deepest gratitude to **Mr. Ravi Shanker** for his guidance, advice, understanding and supervision throughout the development of this dissertation study. I would like to thank to the **Project Approval Committee** members for their valuable comments and discussions. I would also like to thank to **Lovely Professional University** for the support on academic studies and letting me involve in this research.

I express my special thanks to my teachers and friends who have helped me in many ways for the successful completion of this dissertation. All the simple doubts from their part has also made me think deeper and helped me to gain a better understanding about this work.

Last but not the least, thanks to my parents for helping me get where I am today.

Date: **Sumit Kumar**

Reg. No. 11600788

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

With the advent of smartphones, mobile application industry is becoming one of fastest growing industry today. Every now and then, we hear about a new app being launched. However, besides providing you with information like news, fun and amusement services, they can also seize your privacy. One of the most common example of this trend is asking permission from users when they are seeking to download those apps. Many researchers have suggested that, users don't care much while giving permissions to these apps. The main purpose of our research is to know the reason for asking these permission requests by analyzing your app's traffic to detect if there is any leak of private data which is not intended by the user and to know how the app vendors collect sensitive information such as your phone's IMEI number or location for advertisement, tracking, or analytical purposes. In this report, we analyze the mobile apps privacy framework which primarily focuses on the effect of sensitive data leakage and privacy risks involved with it. By distinguishing how and where the information is leaked, the objective is to make the developers aware about such potential vulnerabilities in their applications that can pose threat to user's privacy.

Keywords: *Android Privacy, Mobile Apps, Sensitive Data, Permissions Analysis, Data Flow Analysis, Traffic Analysis, Privacy Leaks, Security Models, Malicious Apps*

CHAPTER 1

INTRODUCTION

1.1 Introduction

Smartphones user base has grown with a rapid rate over the last couple of years. With the emergence in the ownership of smartphones, mobile apps platform became very prominent, that gives users the liberty to download different types of applications from their App Stores[1] ranging from entertainment to work into their mobile phones. Based on a survey done in June of 2016, it has been shown that more than 2,200,000 apps has been provided by the Google's App Store; same is the case with Apple which provides more than 1 million apps in its store. Mobile apps are able to use numerous capabilities of a smartphone ranging from making a simple call to user's location, thereby providing its users with relevant services and striking features.

Admittance to these valuable services and features provides different types of security and privacy invasions which is inescapable. One clear problem is Malwares, other severe issue is that, smartphone handlers, in a generic way, are neither completely conscious and nor have complete control on how these apps gain access and transmits their private information. For example, the KMPlayer app gathers data regarding Device ID & call information which can make users very uncomfortable. In fact, studies [2] [3] have suggested that customers have very little or no understanding regarding these delicate information.

Many studies done by researchers have shown that a stunningly high percentage of mobile applications can access their personal information behind users consent and may threaten their privacy. A recent research [2,3] found out that more than 30 out of 160 Android apps examined and send information related to geographical data to their remote ad servers without user's awareness.

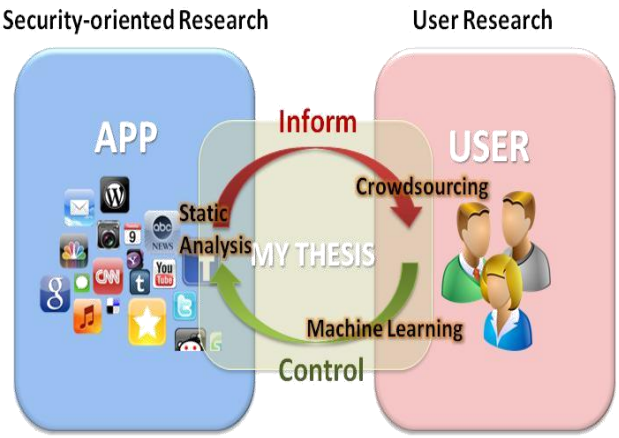

Figure 1.1: Helping users to control their privacy settings without burdening them with numerous decisions [2]

Many of these malicious apps also send the unique mobile IMEI and even the actual cell and serial number to app sellers. All this information helps the advertisers and sellers to make complete and exclusive profiles regarding your interests, likes and dislikes, different locations that you visit while carrying your mobile phone, your social sites surfing habits and much more. One renowned music app was under federal examination [4] for collecting their customer's locality, gender, date of birth, and unique cell phone number (like IMEI) and sends this information to third party servers like advertisers. Social networking applications like Facebook and Path, were being caught time and again, uploading the whole contact lists of their user's onto their servers, which greatly astonished the whole world and questions their trustfulness. Our main work is to provide important and useful information to the end user's such that it will bring down the gap created between user's privacy preferences and research based on privacy.

In this section, a brief description about the current privacy preferences frameworks of the two prevalent mobile operating systems, namely Android and Apple's iOS is provided, in order to get the basic idea about the problems and challenges behind this topic.

1.2 Basic Android Privacy Framework

The privacy preferences framework of Android OS is used to fulfill two purposes in order to protect their customers: one is to restrict the access of smartphone apps to delicate

information (like device ID, system files etc.) and second is to help them in taking proper choices before mounting the apps. Before installing any app from the Google's app store, users are requested to give permission to the app for accessing resources. On the basis of this information users can make the decision on whether to believe that application or not. In order to advance for installation, users are required to grant the permissions requested by the app vendor. And these permissions cannot be withdrawn once granted, unless of course the user uninstalls the app manually.

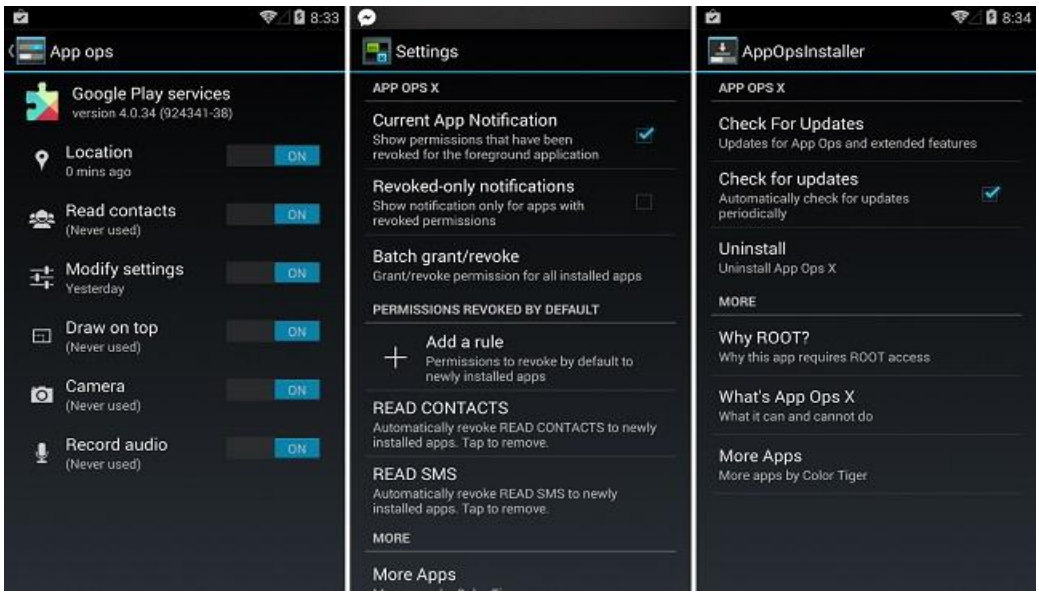

Figure 1.2: "App Ops" – Permissions Screen in Google Play Store[5]

Even though Android's privacy & permission framework is intended to be a more open platform, still it's the duty of their users to make appropriate choices. In spite of that, the ordering of permissions required and their brief description have been improved by Google through the years.

If you look at the current permission screen in android framework, you can only get an abstract view which lacks adequate explanations. Another problem is regarding the lack of control the user have. Previously (Android Jelly Bean), when users grants the permissions to these apps, they have very little control over the how these apps can exploit those permissions other than simply removing it. Hence, it is an all or nothing situation. However, the good news is in the latest versions of android (android 5, 6 & 7), now users

are finally able to make small adjustments to their apps privacy preferences by using a hidden app permission manager also known as “App Ops” feature [2] [6]. The basic idea behind “App Ops”[6] is that if users don’t want any app, say a music app, to gather their personal data like location then they can withdraw that particular permission specifically for that app. However, configuring these applications one at a time seams highly inefficient, since the number of apps and the permissions requested by each of these apps on an average is quite huge, and also sometimes these permission requests are very concise in nature, for instance some of the apps requests the permissions of “Read Phone State and Identity” which is required to mute its own sound (if it’s a music app), but the same permission also allows the access to phone’s unique identity (like IMEI) which can be easily exploited. Hence, bringing the awareness among the users to make proper settings and provide certain level of computerization becomes more and more pertinent.

1.3 Basic Apple-iOS Privacy Framework

The Apple iOS on the other hand implements a different technique, which is actually better compared to what Android has done so far. Apple’s privacy framework doesn’t ask its users to agree for any permissions and also doesn’t provide any information related to data usage. It prompts the users for either accepting or denying any access to sensitive information at the time when that app is used for the first time. The decision thus made is then saved for future references. Similar to “App-Ops” in Android, iOS also provides the liberty to its users to grant access to the sensitive data usage tracking information (like call history, access to file manager, Bluetooth, microphones etc.) for individual application. Still it suffers from a similar kind of usability issue as android because it’s not true for all cases, some permissions are granted automatically without user’s consent.

The motivation behind this research are the given aforementioned issues in these two operating systems and aims to give the control of apps traffic analysis by helping users in terms of better understanding their smartphones and the possible risks of various different apps and their operations.

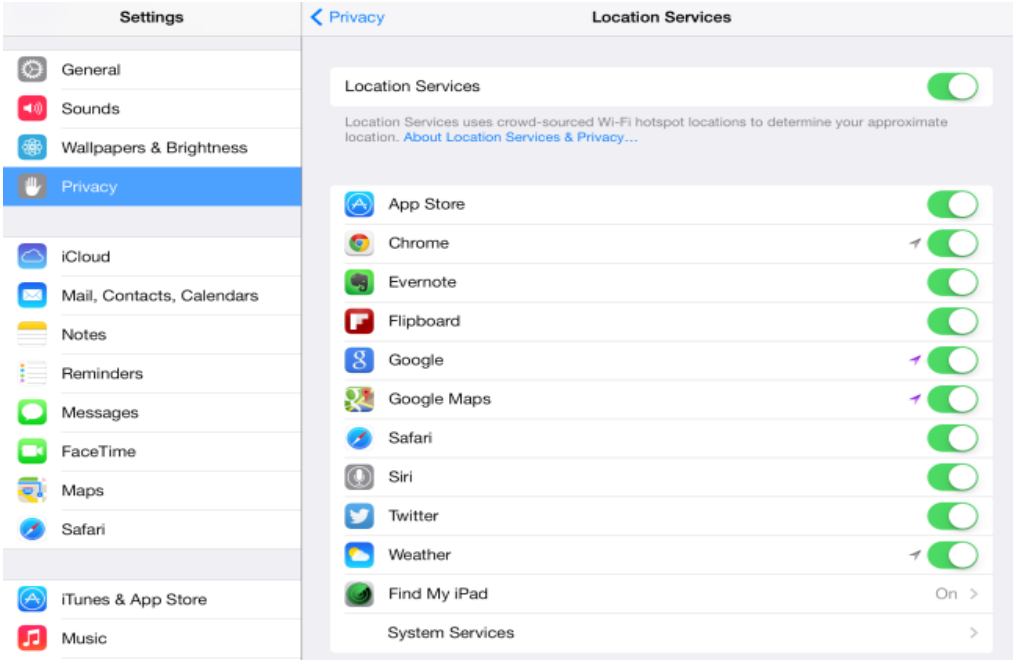

Figure 1.3: Privacy Settings in iOS 6[5]

1.4 Application Analysis Techniques

With advent of internet, protecting user's privacy is the main concern which inspires security researchers to solve this problem and ultimately results in building sophisticated tools to gain a deeper understanding regarding the sensitive behavioral patterns of mobile applications.

In order to keep malicious and harmful apps away from Play Store, Google has introduced a security service codenamed "Bouncer"[7]. This service involves a variety of checks and it runs each app in the background and looks for any suspicious behavior. According to the information published by TrendMicro labs In an effort to drop down the malicious behavior of apps Google scans the entire developers account by themselves, thereby resulting in a 40% reduction of malicious apps in their app store. Hence, this problem is still an open challenge for researchers. In order to detect the privacy leaks in mobile applications, researchers have come up with many useful tools and techniques. These techniques are most preferably used in analyzing apps permissions. Description of these techniques is given below.

Permissions Analysis is the most basic analysis technique [8]. In case of permissions analysis, list of permissions requested by an application during installation or during run time are reviewed and analyzed. Various stages need the application to articulate what features it will try to get access in the midst of execution. The advantage of this approach is that it permits productive survey of thousands of applications in a single time. The limitation is that the review is exactly at an unusual state, without knowing whether the application truly accumulates the requested data and who gets it. There is a connection between number of downloads and the amount of permission requests. The more significant the amount of downloads, the more likely the application requests more approvals. Analyzers may target particular attacks beside these functionalities and try to evade restrictions.

Control Flow Analysis is a technique used to analyze the logical control structures in the application program. Utilizing this technique will enable analyzers to recognize common logical defects, for example, the inability to deal with special cases, and inadequate authorization restrictions. Control flow analysis is expressed using control flow diagrams (CFGs) [9].

Data Flow Analysis technique is one of most basic method to analyze the flow of sensitive data. It searches for different routes among data sources (sensitive data like a file manager database or users contacts lists) and sinks (access points through which our information can leak such as cellular data or Wi-Fi) in the operating systems, since mobile apps runs upon them. At any point if there is a transmission of information from any conceivable source point to sink point without the knowledge of user then it shall be considered as a leak of private data. This technique is especially suited for detecting input-validation errors (like SQL injection attacks) and several private data breaches. Given below table 1.1 lists possible data sources and sinks in a smartphone.

Given below are some common data flow analysis techniques that can be used for detecting privacy leaks.

Sources	Examples
Location Aware Data Sources	Wireless LAN, Last Base Station Location, GPS
Exclusive Identifiers	IMSI, IMEI
Authenticated Data Sources	Cached Usernames & Passwords
Others	Contacts, Calendar, Call log info., Number of Calls etc.

Table 1.1: List of Possible Data Sources[10]

(i) Dynamic Data Flow Analysis

It observes the behavior of mobile apps typically when they are executed in order to detect any privacy leaks. In this technique our main concern is to know how any application program behaves on a sensitive information [10].

Users shall be cautioned regarding some privacy information leak from their mobiles by performing a Dynamic Flow Analysis. To perform this technique, we're required to have an emulator or mobile device with us. However, these Dynamic analysis tools are having several performance related issues like power consumption, processor speed, memory available etc.

One famous Dynamic Analysis tool is Taintdroid[11] which is used to detect the privacy leaks among similar kind of application networks.

(ii) Static Data Flow Analysis

In this technique [12], first the application program code is analyzed statically i.e. without executing it, and then a CFG (control flow graph) is made. This approach attempts to cover different execution paths that can be possible. The transmission of sensitive information from different data sources to possible sink points is traced by the CFG created. The time taken by Static Analysis is more as compared to the Dynamic analysis since all the possible execution paths and complete code is processed by it and hence performance related issues in this technique are less as all the processing is done in a static manner before the actual execution of the code. The static code analysis gives a fully automated scan of mobile applications, and its precision is relied upon the capability of the decompiler as well as the

programming style used by the developer. LinkMiner is one of the famous tool that uses static analysis.

Sink	Examples
Text Messaging	Data sent by simple SMS
File System	Globally readable files can be written by app data
Network	Network access via HTTP or TCP sockets
Intents	Communication among different applications
Others	Usage of 3rd party APIs to access shared memory

Table 1.2: List of Possible Sinks[10]

(iii) Hybrid Data Flow Analysis

This technique [13] selectively make use of both static and dynamic data flow analysis thereby improving the detection of privacy leaks among mobile apps. SmartDroid is an application that uses hybrid technique to detect sensitive data flow. Figure 1.4 shows the main components of static, dynamic and hybrid data flow analysis.

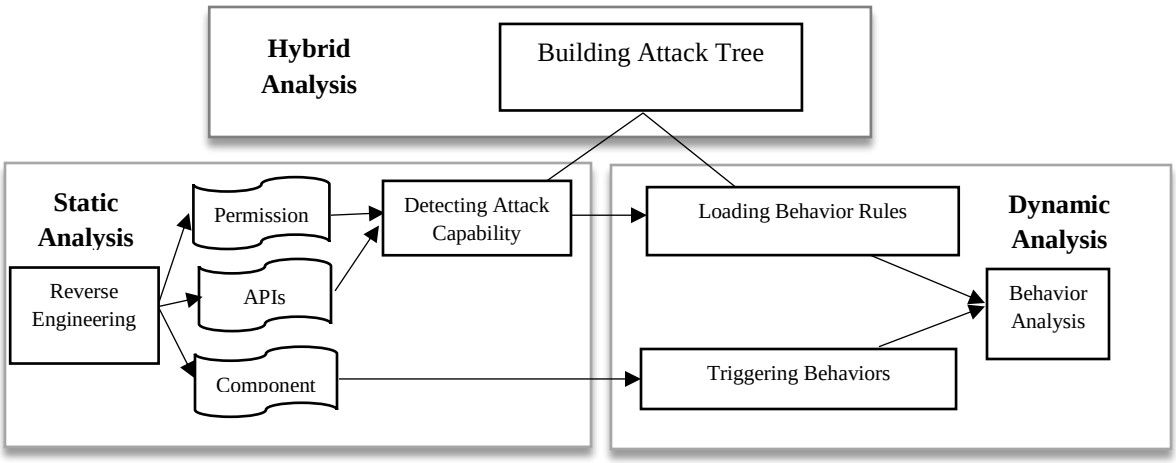

Figure 1.4: Components of Static, Dynamic and Hybrid Analysis

(iv) Cloud Based Data Flow Analysis

Since our smartphones are strictly restricted in terms of resources, because of which it can be a cumbersome to perform privacy detection on these devices. Many researchers have therefore come up with a cloud based flow analysis [14] tools. One of such tool is Paranoid.

It allows a synchronized duplicate model of a mobile phone to run on a cloud based server. Now since, the replica of our mobile phone is on the cloud hence it doesn't have any resource constraints like battery power consumption etc. and this would make it easier to perform privacy leak detection. The emulator that resides on the cloud server is required to perform all privacy checks [15].

1.5 Different Phases of Application Data

There are a few viewpoints that must be considered when expecting to assess and review the security level of an application. Understanding the distinctive states in which the digital information can exists, it can help us to choose the sorts of safety measures and the level encryption that is needed for its protection. In order to perform a thorough privacy and security assessment, it is critical to know all the distinctive states in which data can be found and persists in a mobile device.

There are 3 basic states of application data: data at rest, data currently in use, and data transmitted over the network. Given below, we will give a brief discussion of these three states in which data can exists and in addition the sorts of encryption and security expected to ensure it.

(i). Data at Rest

It refers to the application data that is stored and saved in the storage media of the mobile device. This data is inactive and is currently not being processed or transmitted over the network by the application. Since the data is currently not being acted upon by the application, its state is generally stable. This state of data can be viewed as secure, provided that it is encrypted by using some strong encryption algorithms having a fairly long randomness coupled with a strong key-pair generator.

(ii). Data currently in Use

Data being utilized is the data that isn't just being secured idly on an external or internal storage device. This is the data that is currently being read, processed and modified by the application. It additionally includes the data that is being seen by the users getting it through various application endpoints. Data being utilized is vulnerable against different sorts of risks depending upon where it is in the system and who can use it. This data can be seen as secure, if access to the memory is completely controlled, and if paying little notice to how the application closes, data can't be recuperated from any location other than the data at rest state, which requires re-authorization. The most feeble point for data currently being used is at the endpoints where users can access the application interface and communicate with it.

(iii). Data Transmitted over the network

The third phase is data transmitted over the network. In this phase, application data exits the mobile device over a WiFi access point or GSM network or another mobile device through peer to peer connection like Bluetooth, NFC (Near Field Communication) etc. This information can be considered as secure if the data transmitted over the network is authorized, authenticated and private (encrypted) which implies no unapproved entity can tune in to the communication and eavesdrop the data sent across the network.

CHAPTER 2
REVIEW OF LITERATURE

Nariman Mirzaei et al. (2012) [16] gives an overview on how to address the issues of incompatibility with Java Virtual Machine and path-divergence by enabling the execution of android apps in Java Pathfinder (JPF). This greatly solves the problem of generating test cases for Android apps on JVM rather than on Dalvik Virtual Machine (DVM). Java Symbolic Pathfinder (SPF) is a tool that requires its own virtual machine instead of legacy system (JVM). Therefore, to produce the sample check inputs for mobile apps with the help of SPF, first the java bytecode of the app is generated and then it is executed on JVM.

Secondly, they have leveraged the program analysis techniques to correlate events with their handlers for automatically generating Android-specific drivers that simulate all valid events. They have extended the Symbolic Path Finder and generated the example cases for mobile apps by using the system's call graph model. Their main work highlights on:

(1) Creating stubs that helps them in compilation and execution of Android apps on JPF

(2) To solve the issue related to path-divergence, they have created pseudo classes for Android system library

(3) To mimic the user's behavior by analyzing the systems source code.

Future Work: By applying the appropriate stubs and mock classes they are emerging support for a higher subset of Android system library classes. They are also planning to merge the Symbolic Path Finder with their cloud-based Android testing framework, in order to find out the functional as well as privacy related vulnerabilities.

Their research project has been granted by the Defense Advanced Research Projects Agency.

Grace et al. (2012) [17] have proposed Woodpecker a dynamic flow analysis tool that is used to analyze the capability leaks (i.e. the condition when an app gets access to a permission without invoking any request for it). It examines each application on the device

to investigate reachability of a perilous permission from an open, unguarded interface. This tool categorizes the capability leaks into two types; first is *explicit capability leakage* that permits an application to exploit any publicly available interfaces or services in order to gain access to certain group of permissions without directly requesting for it. Second is, *implicit capability leakage* that instead of exploiting any publicly available service or interface, uses another app to acquire certain permissions having the same certificate authority (CA).

Several security researchers and analysts have utilized different devices from various manufacturers (like Google, Samsung, Motorola and HTC) to identify information leaks by any pre-installed app. And there analysis have uncovered that all pre-installed applications have performed capability leaks.

Yang et al. (2012) [18] in their paper have proposed a static taint analysis approach called Leakminer which helps in detecting the privacy leaks in smartphones. In contrast to dynamic taint analysis, Leakminer detects privacy leakage information on market site before apps are downloaded by the user. Leakminer is executed on the Soot framework, and approximately 1750 android apps are evaluated for any leakage of sensitive data. In their final report they have found out that nearly half of the leakages are true and about 40 false positives were introduced due to the long propagation path inside these applications and due to insufficient context information for the duration of taint analysis.

Finally, they have concluded that their static taint analysis approach took nearly 75 hours to analyze all 1750 apps (2.5 minutes for each app on an average). However, Leakminer introduces no runtime overhead and the execution time can be reduced by further allocating the workload to several machines in a distributed manner.

Zhemin Yang et al. (2013)[19] have proposed an app based validation framework called *AppIntent* which can provide a sequence of events that leads to the data transmission, which ultimately supports the analysts to get to know whether the data transmission is intended by user or not. The basic idea behind this is the concept of event driven symbolic execution. They have also shown the assessment done by *AppIntent* on a set of 750 suspicious apps,

as well as top 1000 free apps available on Google Play store in the year 2013. Their final results have shown the apps which leaks the user's private data.

Future Work: They have also discussed about some of the limitations of this framework like failure in analyzing some apps because their decompilation tool DED. And another limitation is the incompatibility between the native code supported by *AppIntent*.

They have suggested to use a more powerful tool like Dexpler, which can directly parse the DEX (android native code) files without the need for conversion into java bytecode.

Muhammad Haris et al. (2014) [10] provides a detailed survey of mobile computing research, which primarily focuses on privacy risks and data leakage effects. First they have presents an overview of privacy reins implemented in the two major mobile operating systems – Android and iOS. Secondly, they have discussed a number of methods like Data flow analysis (dynamic, static and hybrid), Cloud based analysis along with their practical applications, their architectures and presented different case studies on privacy leaks. They have discussed about the basic mobile cellular technology and the reason of privacy leaks and a short summary of phone sensing and connectivity. Apart from that, they have likewise presented the current research efforts concentrating on refining current techniques to discover and protect sensitive data leaks.

Li Li et al. (2014)[20] have proposed static taint data flow analysis technique that makes use of the control-flow graph (CFG) of applications to discover privacy leaks among apps. They have provided the solution to three difficulties related to internetwork communication (ICC), lifespan of components and callback mechanism creating the CFG imprecision. Their aim is to provide a taint analysis mechanism in order to detect intra and inter-component privacy leaks. They have also claimed that their static taint data flow analysis tool provides more accurate and better results compared to other tools like AppIntent, Woodpecker, CHEX etc. They have taken the help of FlowDroid (a very accurate taint analysis tool used for Android) and Epic (a very effective ICC mapping framework in Android). Finally, they have described the limitations and their solutions of discovering private leaks in Android apps by making an accurate control-flow graph and thereby detecting any inter-component communication based privacy leaks in Android apps.

Rasthofer et al. (2014)[21] proposed a machine learning method of detecting source points and sinks directly in the program of any Android API. Previously, all the research that have been proposed heavily relied upon the manual configuration of lists of source points and sinks which is not easy to obtain. On a given training set of hand-annotated source points and sink points, SuSi (as they named it) identifies the sources and sinks in the entire API and further classifies the source points (e.g., distinctive identifier, location information, etc.) and sink points (e.g., network, file, etc.). In this research paper they have claimed to identify hundreds of source points and sink points with over 92% efficiency, many of which are skipped by the present information-flow tracking devices. This tool adheres supervised learning approach to train the classifier on a comparatively small subset of manually-annotated training examples.

Future Work: Their future work aims to put on this methodology to interfaces for automatically detecting and classifying sensitive flow callbacks. They have also proposed to further investigate on how this method is useful for other settings than Android, e.g., J2EE and various procedural programming languages such as C++, C# or PHP.

Klieber et al. (2014)[22] have proposed a new static taint analysis for Android apps that combines and augments the FlowDroid and Epiee analysis to precisely track both inter-component and intra-component data flow. Two phases are involved for analysis. In the first phase, they have analyzed each app individually. Sources are the received intents, and sinks are considered as sent intents. The phase-1 of their analysis gives the following outcomes:

a) FlowDroid gives the flows within every component,

b) Epiee identifies the properties of the sent intents,

c) Each componets intent filters, which are taken out from the manifest file.

To every location of the source code, an intent ID is given that sends an intent(i.e. the location of source code that calls to a method named *startActivity*). Same ID intents are joined together, while distinct ID intents are termed as distinct sinks.

Using the output given by Phase 1 on a certain number of applications, the analysis of Phase 2 is carried out.

Future work: In order to include the extra data channles like static fields, SQLite databases, and SharedPreferences, the inter-component portion of taint flow analysis is to be enhanced further. They also have envisioned their two phase analysis, which can be further used as: The Phase 1 analyis on each app given in the play store can be executed easily. Whenever any user wants to install any new app from the app store, the user should be told about the possible data flows from that app using the Phase 2 analysis.

Tripp et al. (2014) [23] addressed the privacy implementation problem in mobile apps with a Bayesian classification. The runtime system should be classified as either legitimate or illegitimate based on the sink's behavior whenever the data flow is arrived at the sink point (e.g., any update in database, or any outgoing message). If a release point is valid on the given evidence(that is raised at the sink point), they propose a Bayesian notion of statistical classification. They developed a system called BayesDroid in which an evidence is defined as the likeness among the released data values and personal data held by the device. They have applied the BayesDroid on 54 of the most downloaded apps and thus are able to found out the 27 privacy leaks having only 1 false alarm and also proposed that BayesDroid is substantially more accurate than TaintDroid[16].

Future Work: They have proposed two main objectives for the future course of work. One is to add more features to their tools like accessing modes of the file system (like private or public access), secure HTTP communication and API invocations of privacy significant apps. The other is to apply the static taint analysis to the program code using FlowDroid [24], and thus optimizing the taint based methods for identifying the appropriate values.

Arzt et al. (2014) [24] presents a novel and exceedingly well accurate static taint analysis tool called FLOWDROID. Besides that, they have also proposed a framework called DROIDBENCH which is used for determining the efficiency and accurateness of taint analysis tools for Android based apps. With the help of FLOWDROID they are able to find a large number of privacy data leaks with a very low rate of false positives. FLOWDROID further extends the Soot and Spark framework, in which the former provides the three-address code representation called Jimple and the latter provides an accurate call-graph analysis of the given code. They have used a plugin known as Dexpler which helps FLOWDROID in converting Android's Dalvik bytecode into Jimple.

In the sample test, they have tested a vulnerable android app called InsecureBank and finds seven data leaks with no false positive or false negative report. The analysis takes approximately 30 seconds on a laptop computer having 4 GB of RAM and Intel core 2 Centrino processor with Java Runtime Environment 1.7 as the backend virtual machine. They have also analyzed 500 real-world applications from Google's Play Store, and uncover about 2 leaks per sample on average.

Future Work: As a future work they have planned to improve this tool further by providing the support for handling reflections.

Razaghpanah et al. (2015)[25] have addressed the issues regarding the transparency of operations in android apps in overcoming the barriers to large-scale deployment by employing a unique method that takes the advantage of the VPN API on Android based mobile phones and designed a network flow analysis tool called Haystack, which requires a full access to device's network traffic without the need of any root access privileges. They have implemented the design of Haystack in an Android app which is available on Google Pay Store for free. Besides this, they have also demonstrated the utility of this app to its users and researchers in categorizing mobile app's network traffic and privacy leaks. Haystack uses the VPN API provided by Android that makes a virtual tunnel channel in order to direct all the network data flow traffic into the user space process interface. In order to allow this operation, the user is asked by the client app to permit the request for BIND_VPN_SERVICE, which do not need any rooted device in order to get executed. Secondly, in order to maintain the user space network sockets and to transmit the data traffic through these sockets, it requires the packet header information.

Future Work: Till now, Haystack is being implemented on the Android platform only, and it can be further implemented on other platforms (iOS, Windows, Blackberry etc.) too. Even though Apple's iOS provides strict technical restrictions in application development (for e.g., it doesn't allow TaintDroid to execute on their platform), Haystack's implementation doesn't require any of those permissions that are required by TaintDroid like tools and this allows it to be used on such platforms as well. In order to access the mobiles data traffic we can further explore Haystack by splitting the Traffic analyzer and Forwarder modules. In a further improvement process, they are planning to provide

Haystack's code in an open source platform (like GitHub) and are also planning to make the information collected by Haystack to be made available in a web based platform interface similar to Recon and Censys.io.

Li et al. (2015)[26] proposed a static taint analyzer tool that helps in detecting the leak of private data among different modules of any Android application. IccTa (Inter Component Communication Taint Analysis Tool) supports inter-component detection and greatly improves the analysis precision. They have detected 534 ICC outflows in 108 mobile applications from MalGenome (Android Malware Genome Project) and nearly 2,395 ICC outflows of data in 337 applications from Play Store. In order to detect private data leaks based on Inter Component Communication, they have developed 22 apps and added these apps to DroidBench, which is a framework or testing suite for evaluation of usefulness and accuracy of taint data flow analysis tools. When they have executed IccTa on DroidBench, they showed that it gives a precision of 96.6 % while on MalGenome, it shows 534 ICC leaks with 8.6% accuracy. Their project was supported by a Google Faculty Research Program.

Bosu et al. (2016)[27] presented DIALDroid, a scalable and precise tool in order to analyze inter-app Inter-Component Communication (ICC) among Android apps, which outperforms current state-of-the-art ICC analysis tools. Using DialDroid, they have performed the first large set of discovery of malicious and susceptible apps built on inter-app ICC data flows between 110,150 popular applications and identified key security insights.

The workflow of DialDroid involves four key operations as follows:

a) ICC Entry and Exit Points: For a particular app, they have extracted the attributes and permissions list of the intent filters present in the *AndroidManifes.xml* file. They have performed static analysis using their custom ICC extractor to know the characteristics of the intents going through ICC exit or sink points.
b) DATAFLOW ANALYSIS: They have used the static taint analysis in order to determine the ICC entry and exit leaks in an app.

c) DATA AGGREGATION: They have aggregated the data pulled out in the earlier two steps to insert in a relational database server.
d) ICC LEAK CALCULATION: With the use of SQL queries and stored procedures, they have computed Inter Component Communication with collusive information leakage and privilege increases based on fine-grained security policies.

The time complexity of DialDroid is O(N), where the 'N' represents the total no. of applications that are being analyzed. And, the complexity for calculating the ICC leaks is O(mN), where the 'm' represents the number of apps with leaks at the ICC exit points. Worst case scenario is when m = N. In practice, m would be much smaller than N.

Raul Herbster et al. (2016)[28] have proposed a framework model that avoids the relay of sensitive information to suspicious third party server called *Privacy Capsules (PCs).* Privacy Capsules executes in two phases. First is *unsealed phase,* the app has complete access to the untrusted network resources and second one is *sealed phase*, in which the untrusted app cannot be able to communicate with the untrusted resources but can have access to private input values. Once the app switched to the sealed state, it can only be terminated or restarted but cannot be returned back to the unsealed state. Privacy Capsules provides further mechanisms to allow legitimate flow of private data without compromising user's privacy, these are as follows:

- *Sealed Repository (SR)* → Using this, an application which is in its *sealed state* can collect confidential data continuously during executions. Information thus inserted into the SR is encrypted with a key (which is same for both user as well as the application) which is remain confidential and cannot be leaked by the application. A Sealed Repository could exist on the providers cloud service which can be untrusted in nature.
- *Sealed Channel (SC)* → Using this, live video/voice/chatting data on different users devices could be replaced between instances of an app in a sealed state. The information stored in the *Sealed Channel* is encoded with the keys and the participating users are authenticated. This confidential information is never exposed to the app.

- *Selective Declassification* → Using this feature, with user's consent that particular information could be unveiled to a third party server.

Using these three mechanisms, they have designed three PC-compliant apps prototype model. First is a *train scheduling and ticketing* app which shows transportation schedules, maps, product lists, and permit users to simply browse or buy the products. Second is a *Wellness* app that monitors the health of an individual by communicating to the fitness sensors thereby examine and tabularize the data thus recorded and provides a facility to share the results with family, coaches or health care specialists. Finally, they have presented a *live chat* application, which provides the class of apps that is primarily focused on communication with other users. They have concluded that the prototypes thus implemented have a low battery and performance overhead and is appropriate for a large set of applications for preventing any privacy leaks.

Several tools and frameworks have been proposed in the past to detect privacy leaks from android devices. These tools follows various approaches such as static, dynamic, hybrid, permissions, and cloud based analysis etc. to detect the privacy related behavior of mobile apps. In this section, we have featured some of the frameworks and tools that are used for detecting possible privacy leaks in android apps. Furthermore, we have compared all these tools and provided the result in a tabular format given in table 2.1.

Framework/Tools Used	Analysis Technique	No. of Apps Tested	Year	Results Summary
TISSA [29]	Dynamic Flow Analysis	24	2010	It was discovered that location info. was leaked by 14 apps and phone's IMEI was leaked by 13 apps.
PCLeaks [30]	Static Flow Analysis (Between Components)	2000	2010	About 986 component leaks were detected. Out of which, 534 leaks were due to activity launch, 245 leaks due to broadcast injection and 110 leaks due to activity hijacking. Furthermore, there were 64 leaks due to service launch.
TaintDroid [31]	Dynamic Flow Analysis	30	2010	They have discovered 68 potential instances of privacy leakage through 20 applications.
ScanDal [32]	Static Flow Analysis	90	2011	6 apps were found to be leaking location info. to ad-servers and 5 apps were leaking the same to analytics servers.
Woodpecker [17]	Dynamic Flow Analysis (Capability Leaks)	953	2012	Several pre-installed apps were found vulnerable of capability leaks that includes devices from Google, Samsung, Motorola, HTC etc.
LeakMiner [18]	Static Flow Analysis	1750	2012	127 apps were found to be leaking device IMEI, 12 apps leaks contact details, and 27 apps leaks users location data.
AndroidLeaks [33]	Static Flow Analysis	25,976	2012	A total of 57,299 leaks were found; out of which 65.51% of the leaks are for advertisement purpose, 92% of the leaks are linked with phone data, 5.94% related to location, and 0.61% leaks of audio data.
CHEX [34]	Static Flow Analysis (Between Components)	5,486	2012	Out of 5,486 android apps, they have found out 254 potential vulnerabilities related to component hijacking.
SmartDroid [35]	Hybrid Analysis	--	2012	It helps in detecting automatic UI based trigger conditions which are required to reveal the malicious behavior in android apps.
AppIntent [19]	Static Flow Analysis	1,000	2013	140 apps were found to have the potential data breach, out of which 26 apps were leaking data without users consent, 24 apps leaks phone's IMEI number, 1 app each leaks contacts and messages info to third party servers.

Framework/Tools Used	Analysis Technique	No. of Apps Tested	Year	Results Summary
VetDroid [36]	Dynamic Flow Analysis (Permissions Analysis)	1,249	2013	It highlights an efficient system to build a tool based on the permissions behavior of android apps. This eases malware detection and helps in finding vulnerabilities that are otherwise hard to find.
DroidTest [37]	Dynamic Flow Analysis	50	2013	They have discovered that most of the apps leaks data related to device IMEI, IMSI (Subscriber ID), and users phone number.
IntentFuzzer [38]	Dynamic Flow Analysis (Capability Leaks)	2,183	2014	It is discovered that more than half of the applications are vulnerable to capability leaks related to phone and network state, location and connectivity.
FlowDroid [24]	Static Flow Analysis	500	2014	It performs data leakage detection with a precision of 86% and greatly outperforms some of the famous commercial tools. It achieves a recall rate of 96% with only 9 false positives.
AmanDroid [39]	Static Flow Analysis	753	2014	It successfully detects different types of data leaks like Passwords, OAuth tokens, API misuse etc. And outperforms FlowDroid in terms of positive test results.
A5 [40]	Hybrid Analysis	1,260	2014	Mainly used to detect malicious network activity and capturing network threats with a processing time of 149 seconds per sample.
AMDetector [41]	Hybrid Analysis	728	2014	It is an automated detection tool for detecting malware apps on Google play store with a success rate of 88.14% and false positive rate of less than 1.8%.
Andrubis [42]	Hybrid Analysis	1,000,000	2014	It has analyzed over 91.67% of all the apps successfully out of which, 0.34% were failed due to bugs and 7.99% of the samples are having a corrupt API.

Framework/Tools Used	Analysis Technique	No. of Apps Tested	Year	Results Summary
CoChecker [43]	Static Flow Analysis	1,123	2014	It identifies the potential leak paths to detect privilege escalation by malicious apps. It raises alarm for 117 potential data leaks among which 84 were capability leaks and 33 were private data leaks.
DroidTrace [44]	Dynamic Flow Analysis	50,000	2014	It detects malicious apps using ptrace dynamic analysis and classifies app behaviors on the basis of file access, IPC (inter-process communication), network connection and privilege escalation.
Gort [45]	Hybrid Analysis and Crowd Analysis	40	2014	It is a heuristic framework that helps security researchers and analysts to detect privacy leaks and privacy behavior of android apps.
IccTa [26]	Static Flow Analysis (Between Components)	3,000	2015	425 applications leaks the information related to device ID and location of the user without users consent.
MARVIN [46]	Hybrid Analysis	135,000	2015	This tool leverages machine learning techniques to analyze the potential privacy risks associated with any android application in the form of malice score. It successfully classifies 98.24% of the malicious apps with false rate as low as 0.04%.
Privacy Capsules [28]	Dynamic Flow Analysis	--	2016	It is an execution framework for android apps to detect potential privacy leaks. It has a lower processing overhead and capable of executing large class of applications.
DialDroid [27]	Dynamic Flow Analysis	100,206	2017	It performs inter-app flow analysis on such a huge dataset within a reasonable time and identified vulnerable apps based on ICC (Inter Component Communication leaks).
POSTER [47]	Static Flow Analysis (Between Components)	08	2017	This tool analyzes multiple apps simultaneously and detects potential privacy leakage during inter app communication and executed their data set on DroidBench(a testing framework).

Table 2.1: A comparison among different privacy frameworks

CHAPTER 3
PROBLEM DEFINITION

3.1 Privacy: A Fundamental Right

Over the previous decade, security has increased huge consideration in the scholarly world and in addition to the industry. The fundamental purpose for this intrigue is the result of privacy infringement on people. From one perspective, user's personal information can be abused by pernicious identities to take or uncover individual data about the user and then again it can be abused to hurt users monetarily or socially. Besides, organizations can likewise utilize this information to learn delicate individual identifiable data about users without their assent and mindfulness.

In spite of the fact that subtle information of private data can fluctuate, implications of protection have similarities crosswise over various issues. Numerous definitions have been proposed by research scholars to comprehend the social importance of privacy and security [48]. Yet, the definition of privacy changes based on individuals preferences. The amount of data that one user is ready to disclose may differ from the amount of data that a more conscious user is ready to give up.

Lin et al. [49] have characterize the users based on their privacy preferences into 4 different categories:

(i). Conservatives (11.9%): This kind of users feels extremely awkward in letting their delicate individual data (for example, phone IMEI number, location etc.) be utilized by 3rd party app vendors. They likewise feel uncomfortable if these apps utilizes their location info, unique ID, call log info/saved contacts or messages internally (for their core functionality) if the need of utilizing these sensitive individual information is not noticeable to them.

(ii). Unconcerned (23.34%): This category belongs to the users who are comfortable in revealing their delicate information (nearly) for each situation, regardless of who is gathering their information and for what purposes.

(iii). Fence-Sitters (50%): Most users inside this group don't have a solid opinion for the usage of different permissions by mobile apps. These people feels extremely comfortable in letting mobile apps get to their private information for their own functionality.

(iv). Advanced Users (17.95%): These are called advanced because they have a more nuanced comprehension of delicate information utilizations. It means that, these users have knowledge about what kind of information is used by the apps and for what purpose (like for apps core functionality).

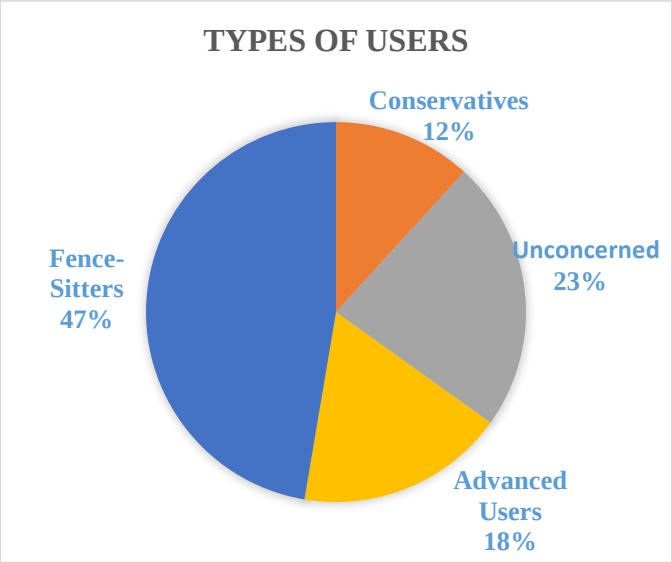

Figure 3.1: Types of users according to their privacy behaviors

3.2 App Permissions: A Dilemma

The permission model of Android apps is multifaceted [50]. The permissions can be asked for accessing the interface (API), Accessing the file system, and for Inter-Process communication. Some of these high-level permissions often maps to the lower-level OS kernel. For example, accessing Bluetooth service, opening sockets, and accessing certain file system paths which are otherwise not allowed.

Android processes these app permissions in AndroidManifest.xml file. This application manifest is presumably the most critical source of data for Android application security experts. It contains the majority of the information in regards to an application's permission and gives us a very few insights about how apps components will be permitted to interact with the remaining components of the application.

3.3 Issues with Apps Permission List

Starting in Android 6.0 (API 23), users can allow the applications to access certain resources while it is running, and not at the time of installation (As with API 5 and below). It additionally gives the users more control over the applications usefulness; for instance, a user can give a music app access to its microphone but not to its location at the same time. However, Android app developers can ask for more number of permissions for their apps core functionality than it might requires [51]. The explanation for why the app requires certain permissions which might be dangerous in terms of user's privacy depends upon the app developers. In case if a user wants to use the functionality that needs a certain permission (like asking for users location or accessing contacts list), which the user is not comfortable with and hence continues turning down the permission request, that most likely demonstrates that the user doesn't comprehend why this application needs that permissions to acquire the required functionality.

In other case, suppose a legitimate messaging app wants to read the contacts list, which the user grants access to, still there is no full proof guarantee that the app developer will not use this information in any harmful manner like sending it to advertising companies. These sorts of oversights, which appear harmless, frequently prompt bad practices from developer's side like under-granting or over-granting the permissions. In case if it's under-granting, it's often a functionality or reliability issue, for instance, an unhandled security exception prompts the application to crash.

With respect to over-granting, it's increasingly a security issue, for instance, an over-privileged app exploits the users privacy.

3.4 Lack of Transport Security

Apps which uses permissions to access the internet will make use of mobile data, or Wi-Fi to transmit the sensitive data. It is the responsibility of app developers to ensure the confidentiality, integrity, and non-repudiation of data. However, because of the absence of comprehension about how to correctly implement Secure Socket Layer (SSL) [52] or Transport Layer Security (TLS), or simply the inaccurate thought that "if it's over the

internet, it's protected", application developers neglect to ensure the safety of data that is transmitted.

This issue tends to show in at least one of the given ways:

- Use of weak encryption algorithms or no encryption at all
- Encryption is strong but lack of security warnings or no certificate validation
- Improper use of transport security according to given network type (for instance, mobile data vs. Wi-Fi)

3.5 Lack of Database security

Android offers various standard methods for storing user's data – namely SQLite databases, Shared Preferences, and plain files. Different applications (based on their functionalities, stores user's sensitive data like passwords, authentication tokens (like OTPs), Contacts, Communication records (like call logs), website names used by users for sensitive services (like social media, personal blogs, health and dating websites). These applications saves the user data in databases, XML files etc. It's important to assess the security of these storage mediums.

The most common mistakes that app developers do is storing the sensitive data in plaintext format, using unprotected structured interfaces like content providers and unreliable file authorizations (permissions to read, write and execute).

3.6 Neglecting IPC Endpoint Security

The basic inter-process communication endpoints (IPC), which includes – Activities, Services, Content Providers and Broadcast Receivers, are regularly neglected as potential attack vectors. At its most essential level, security of these interfaces is ordinarily accomplished by requesting app permissions. For instance, an application may characterize an IPC endpoint that ought to be accessible by other components of the same application or by different application that requests the same permission. In the latter case, any malicious app can access the unsecure endpoints and thus be able to gather user's private data without users knowledge.

CHAPTER 4

SCOPE OF THE STUDY

With the increase in smartphone's usage, privacy of the user's has become the primary concern of researchers working in the field of security. Smartphone apps are prevalently being used these days for leisure and fun activities, work related, news and so on. Before installing any of these apps, they generally asks of permissions to access different components of your device like calls, media, microphone, cell ID, Wi-Fi and Bluetooth information etc. for their proper functioning.

However, sometimes these apps requests for unnecessary permissions for example a music app might try to access your contacts list or a simple gaming app wants to access your location or media files. Users generally don't think much before accepting those permissions, one reason for that could be – they don't have any option except to accept those permissions since there is no choice provided in Google Play Store to select which permissions user allow and which are not allowed. Because of this disagreement, some malicious apps might use these permissions to exploit user's privacy by selling their information to ad servers or for blackmailing users by leaking their private data. Recently, WhatsApp is being sued in the Delhi High Court for sharing its user's data (like names, chat logs, device status, connection information, phone numbers etc.) to Facebook (its parent company) without user's knowledge.

To address this issue, many research have been done from the day smartphone markets are boomed. Researchers have used different data flow analysis techniques to analyze the app's traffic and detect any privacy leaks and found out that almost 20-30% of the apps leaks private data to third party servers. However, they provide any detailed analysis of these leaks and their results are not very transparent to the users.

Our aim is to work on these drawbacks and show the true nature of such malicious apps by answering the questions on how these apps use these permissions to exploit the user's privacy for their own benefit.

CHAPTER 5

OBJECTIVES OF THE STUDY

Our main objective is to assess the privacy and security of mobile applications particularly for android platform since it is open source, popular and widely used platform. We began with examining and distinguishing each conceivable state at which the data can exist, which is the essential prerequisite, keeping in mind the end goal is to have the capacity to keep the data safe and secure. Furthermore, to provide useful information to end user's by analyzing mobile apps traffic and revealing how these apps communicate with tracking services and how they collect sensitive personal information for their gain and to show the result of our research in a single open platform to all its user's.

Many research have suggested that, users don't care much while giving permissions to these apps. The main purpose of our research is to know the reason for asking these permission requests by analyzing your app's traffic to detect if there is any leak of private data which is not intended by the user and to know how the app vendors collect sensitive information such as your phone's IMEI number or location for advertisement, tracking, or analytical purposes.

We have consolidate our research objectives into three research questions which are shown below:

(i). Where the data can exists?

(ii). How personal data is handled by the applications?

(iii). How can one properly assess the privacy and security of mobile applications?

Meanwhile, we acknowledge that privacy has many faces. Our work will only point out some of the ways and techniques to tackle this problem. Increasing awareness among consumers and application developers, implementing IT laws and guidelines are also some other aspects, which are crucial for the protection of smartphones user's privacy.

CHAPTER 6

PROPOSED RESEARCH METHODOLOGY

Based on our research objectives we have proposed the following work plan:

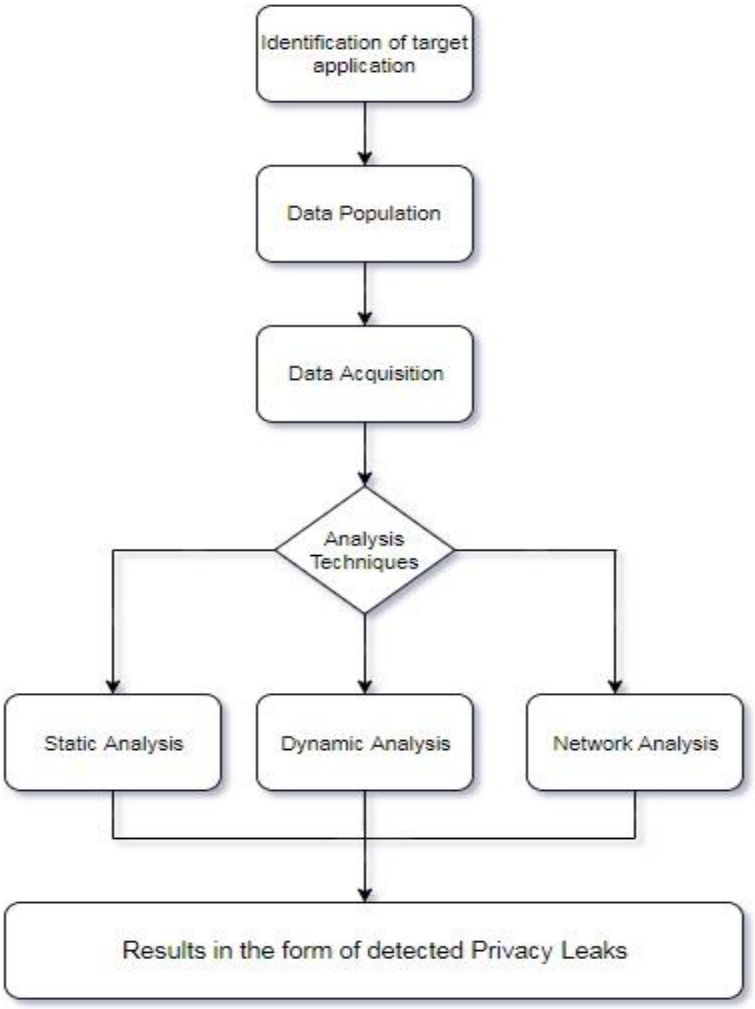

Figure 6.1: Steps in the process of privacy leaks detection

A. Identification of the target application

Among several categories, like social media, banking, games, productivity, e-commerce etc. we will take some popular free apps (based on their number of downloads) that are available on Google play store from each category.

B. Data Population

Next step is to populate the database of these applications with enough data such that we can perform our in-depth analysis. We will create dummy user accounts and assess the application for potential vulnerabilities in terms privacy leakage.

C. Data Acquisition

We are going to perform all these tests on android device emulator. To acquire the required data, we will use the Android Debug Bridge (ADB) which is used to fetch the entire application database into the host machine (including applications APK file).

D. Analysis Techniques

(i). Static Analysis

This technique will be used to assess the applications data when it is not currently in use. Data at rest phase comes under static analysis. Given below is the table that describes different vulnerability types and tools used for its assessment.

(ii). Dynamic Analysis

This technique is used when the application is executed that is, when the application data is currently in usage. Several vulnerability types (under dynamic analysis) are discussed in the table given along with the tool that will be used for its assessment.

(iii). Network Analysis

This technique will find the potential leaks when the user sends any sensitive data over the network. Several vulnerability types (under network analysis) are discussed in the table given along with the tool that will be used for its assessment.

E. Final Results

In the final step, the obtained results in the form of number of privacy leaks detected, type of information being disclosed, application vulnerabilities etc. will be discussed.

Static Analysis		Dynamic Analysis		Network Analysis	
Vulnerability	**Tools Used**	**Vulnerability**	**Tools Used**	**Vulnerability**	**Tools Used**
Validation of app signatures	Keytool	Memory dump analysis	LiME/DMD	HTTP traffic inspection	Burp Suite
Inspection of APK file	Androguard/ CodeInspect	Exploitation of race conditions	--	Validation of SSL certificates	Wireshark/ NetworkMiner
Insecure data storage	ADB	Stack memory corruption	LiME	App sync procedure	--
Information log disclosure	Logcat	Insecure intent communication	Drozer/ComDroid	Usage of 3rd party APIs	Gort/Haystack

Table 6.1: Categorization of different analysis techniques, their vulnerabilities and tools used

CHAPTER 7
EXPECTED OUTCOMES

Our expected outcomes includes the comprehensive assessment of privacy and security of mobile applications along with the identification of vulnerable apps and potential privacy leaks that can be possible during different phases of application data.

Secondly, the implementation of customized security profiles for each and every user based on their priorities and privacy concerns on how much information the user is willing to give to the requesting application, thereby making users aware about how their private data can be utilized by different third party applications and organizations without their consent. We will also conduct some research surveys to help users in identifying permission usages by applications and isolate them based on their permission requests along with the purpose associated with that permission. Thereby, making a small contribution in the area of privacy and security of users.

CHAPTER 8
SUMMARY & CONCLUSIONS

In this study, we have given a detailed and wide-ranging overview of the recent research contemplates on the privacy and security frameworks of mobile applications [53]. Trust and Privacy being the most important topic of discussions in today's world, this paper provides some insights about how user's personal data can be leveraged by some malicious mobile applications for their own benefits. In the first section we have presented a brief introduction about the Android Apps privacy frameworks, we have given the categorization of users on the basis of their privacy behaviors. Moreover, we have discussed some of the analysis techniques like data flow analysis, permissions analysis, and crowd analysis, that can be used to identify potential privacy leaks. Then we discussed the literature review where we have given a brief overview of the past research work done in the area of privacy and security of android applications. Along with that, we reviewed some of the privacy frameworks and tools proposed by security researchers and analysts in the past and provided a detailed comparison table of the frameworks developed in the past along with their results summary. In the Problem definition section we have talked about some of the potential weaknesses in Androids privacy model for instance, lack of transport layer security and database security which can be exploited by any unknown app and results in possible leakage of sensitive information. Then we discussed about the scope and objectives of our research work by consolidating it into three basic research questions that needs to be answered. In the research methodology section we have presented the proposed work plan for every conceivable state of data based on different analysis techniques along with the tools that we are planning to use. Finally, we have discussed the possible research outcomes that we expect from our study on privacy and security of android applications.

Based on our survey and the results thus obtained from the past research, we have plotted the graph between No. of Apps leaking sensitive information vs Type of information leaked. However, this graph is not very precise (in numbers) and is based upon certain

observations and trends from the past results and gives an idea about what type of information is most notably being used by these applications.

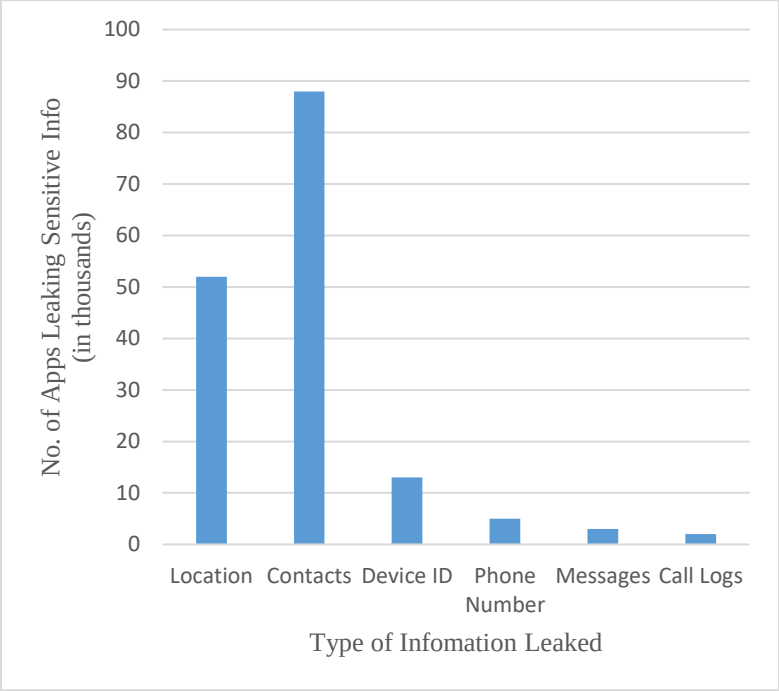

Figure 8.1: Trends showing what kind of information is leaked

REFERENCES

[1] Number of Apps available in leading App Stores as of June 2016. [Online] Available: https://www.statista.com/statistics/276623/number-of-apps-available-in-leading-app-stores/

[2] Lin J. "Understanding and capturing people's mobile app privacy preferences" Carnegie-Mellon University Pittsburgh PA School of Computer Science; 2013 oct 28.

[3] Sadeh, J. L. B. L. N., and Jason I. Hong. "Modeling users' mobile app privacy preferences: Restoring usability in a sea of permission settings." *Symposium on Usable Privacy and Security (SOUPS)*. Vol. 40. 2014.

[4] Federal Trade Commision. Mobile privacy disclosures: building trust through trasparency. Federal Trade Commision 2013. [Online] Available http://www.ftc.gov/os/2013/02/130201mobileprivacyreport.pdf

[5] Kang, Jina, Hyoungshick Kim, Yun Gyung Cheong, and Jun Ho Huh. "Visualizing Privacy Risks of Mobile Applications through a Privacy Meter." In Information Security Practice and Experience, pp. 548-558 Springer International Publishing, 2015.

[6] "App Ops – What you need to know", Android Authority. December 2013 [Online]. Available: http://www.androidauthority.com/app-ops-need- know-324850/

[7] "A look at Google Bouncer", TrendLabs Security Intellegence Blog. July 2012. [Online]. Available: http://blog.trendmicro.com/trendlabs-security-intelligence/a-look-at-google-bouncer/

[8] Felt, Adrienne Porter, Erika Chin, Steve Hanna, Dawn Song, and David Wagner. "Android permissions demystified." In Proceedings of the 18th ACM conference on Computer and communications security, pp. 627-638. ACM, 2011.

[9] Barrera, David, H. Gunes Kayacik, Paul C. van Oorschot, and Anil Somayaji. "A methodology for empirical analysis of permission-based security models and its application to android." In Proceedings of the 17th ACM conference on Computer and communications security, pp. 73-84. ACM, 2010.

[10] Haris, Muhammad, Hamed Haddadi, and Pan Hui. "Privacy leakage in mobile computing: Tools, methods, and characteristics." arXiv preprint arXiv:1410.4978 (2014).

[11] Chun, S. Han V. Tendulkar B G, LP Cox J. Jung P. McDaniel, A N Sheth W. Enck, and P. Gilbert. " TaintDroid: an information - flow tracking system for realtime privacy monitoring on smartphones. " ACM Transactions on Computer Systems (TOCS) (2014).

[12] Lokhande, Bhushan, and Sunita Dhavale. “Overview of information flow tracking techniques based on taint analysis for android.” In Computing for Sustainable Glabal Development (INDIACom), 2014 International Conference on, pp. 749-753. IEEE, 2014.

[13] Sarwar, Golam, Olivier Mehani, Roksana Boreli, and Mohamed Ali Kaafar. “On the Effectiveness of Dynamic Taint Analysis for Protecting against Private Information Leaks on Android-based Devices.” In SECRYPT, pp. 461-468. 2013.

[14] Mahmood, Riyadh, Naeem Esfahani, Thabet Kacem, Nariman Mirzaei, Sam Malek, and Angelos Stavrou. “A whitebox approach for automated security testing of Android applications on the cloud.” In Automation of Software Test (AST), 2012 7th International Workshop on, pp. 22-28. IEEE, 2012.

[15] Burguera, Iker, Urko Zurutuza, and Simin Nadjm-Tehrani. “Crowdroid: behavior-based malware detection system for android/” In Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices, pp. 15-26. ACM, 2011.

[16] Mirzaei, Nariman, et al. "Testing android apps through symbolic execution." *ACM SIGSOFT Software Engineering Notes* 37.6 (2012): 1-5.

[17] Grace, Michael C., Yajin Zhou, Zhi Wang, and Xuxian Jiang. “Systematic Detection of Capability Leaks in Stock Android Smartphones.” In NDSS, vol. 14, p. 19. 2012.

[18] Yang, Zhemin, and Min Yang. "Leakminer: Detect information leakage on android with static taint analysis." In *Software Engineering (WCSE), 2012 Third World Congress on*, pp. 101-104. IEEE, 2012.

[19] Yang, Zhemin, et al. "Appintent: Analyzing sensitive data transmission in android for privacy leakage detection." *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*. ACM, 2013.

[20] Li, Li, Alexandre Bartel, Jacques Klein, and Yves Le Traon. "Detecting privacy leaks in Android Apps." (2014).

[21] Rasthofer, Siegfried, Steven Arzt, and Eric Bodden. "A Machine-learning Approach for Classifying and Categorizing Android Sources and Sinks." *NDSS*. 2014.

[22] Klieber, William, et al. "Android taint flow analysis for app sets." *Proceedings of the 3rd ACM SIGPLAN International Workshop on the State of the Art in Java Program Analysis*. ACM, 2014.

[23] Tripp, Omer, and Julia Rubin. "A Bayesian Approach to Privacy Enforcement in Smartphones." In *USENIX Security*, vol. 14, pp. 175-190. 2014.

[24] Arzt, Steven, Siegfried Rasthofer, Christian Fritz, Eric Bodden, Alexandre Bartel, Jacques Klein, Yves Le Traon, Damien Octeau, and Patrick McDaniel. "Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps." *Acm Sigplan Notices* 49, no. 6 (2014): 259-269.

[25] Razaghpanah, Abbas, Narseo Vallina-Rodriguez, Srikanth Sundaresan, Christian Kreibich, Phillipa Gill, Mark Allman, and Vern Paxson. "Haystack: A Multi-Purpose Mobile Vantage Point in User Space." *arXiv preprint arXiv:1510.01419* (2015).

[26] Li, Li, Alexandre Bartel, Tegawendé F. Bissyandé, Jacques Klein, Yves Le Traon, Steven Arzt, Siegfried Rasthofer, Eric Bodden, Damien Octeau, and Patrick McDaniel. "Iccta: Detecting inter-component privacy leaks in android apps." In *Proceedings of the 37th International Conference on Software Engineering-Volume 1*, pp. 280-291. IEEE Press, 2015.

[27] Bosu, Amiangshu, Fang Liu, Danfeng Daphne Yao, and Gang Wang. "Android Collusive Data Leaks with Flow-sensitive DIALDroid Dataset."

[28] Herbster, Raul, Scott DellaTorre, Peter Druschel, and Bobby Bhattacharjee. "Privacy capsules: Preventing information leaks by mobile apps." In *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services*, pp. 399-411. ACM, 2016.

[29] Zhou, Yajin, Xinwen Zhang, Xuxian Jiang, and Vincent W. Freeh. "Taming information-stealing smartphone applications (on android)." In International conference on Trust and trustworthy computing, pp. 93-107. Springer, Berlin, Heidelberg, 2011.

[30] Li, Li, Alexandre Bartel, Jacques Klein, and Yves Le Traon. "Automatically exploiting potential component leaks in android applications." In Trust, Security and Privacy in Computing and Communications (TrustCom), 2014 IEEE 13th International Conference on, pp. 388-397. IEEE, 2014.

[31] Enck, William, Peter Gilbert, Seungyop Han, Vasant Tendulkar, Byung-Gon Chun, Landon P. Cox, Jaeyeon Jung, Patrick McDaniel, and Anmol N. Sheth. "TaintDroid: an information-flow tracking system for realtime privacy monitoring on smartphones." ACM Transactions on Computer Systems (TOCS) 32, no. 2 (2014): 5.

[32] Kim, Jinyung, Yongho Yoon, Kwangkeun Yi, Junbum Shin, and S. W. R.D. Center. "ScanDal: Static analyzer for detecting privacy leaks in android applications." MoST 12 (2012).

[33] Gibler, Clint, Jonathan Crussell, Jeremy Erickson, and Hao Chen. "AndroidLeaks: Automatically Detecting Potential Privacy Leaks in Android Applications on a Large Scale." Trust 12 (2012): 291-307.

[34] Lu, Long, Zhichun Li, Zhenyu Wu, Wenke Lee, and Guofei Jiang. "Chex: statically vetting android apps for component hijacking vulnerabilities." In Proceedings of the 2012 ACM conference on Computer and communications security, pp. 229-240. ACM, 2012.

[35] Zheng, Cong, Shixiong Zhu, Shuaifu Dai, Guofei Gu, Xiaorui Gong, Xinhui Han, and Wei Zou. “Smartdroid: an automatic system for revealing ui-based trigger conditions in android applications.” In Proceedings of the second ACM workshop on Security and privacy in smartphones and mobile devices, pp. 93-104. ACM, 2012.

[36] Zhang, Yuan, Min Yang, Bingquan Xu, Zhemin Yang, Guogei Gu, Peng Ning, X. Sean Wang, and Binyu Zang. “Vetting undesirable behaviors in android apps with permission use analysis.” In Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security, pp. 611-622. ACM, 2013.

[37] Rumee, Sarker T. Ahmed, and Donggang Liu. “DroidTest: Testing Android applications for leakage of private information.” In Information Security, pp. 341-353. Springer International Publishing, 2015.

[38] Yang, Kun, Jianwei Zhuge, Yongke Wang, Lujue Zhou, and Haixin Duan. “IntentFuzzer: detecting capability leaks of android applications.” In Proceedings of the 9th ACM symposium on Information, computer and communications security, pp. 531-536. ACM, 2014.

[39] Wei, Fengguo, Sankardas Roy, and Xinming Ou. “Amandroid: A precise and general inter-component data flow analysis framework for security vetting of android apps.” In Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, pp. 1329-1341. ACM, 2014.

[40] Vidas, Timothy, Jiraqi Tan, Jay Nahata, Chaur Lih Tan, Nicolas Christin, and Patrick Tague. “A5: Automated analysis of adversarial android applications.” In Proceedings of the 4th ACM Workshop on Security and Privacy in Smartphones & Mobile Devices, pp. 39-50. ACM, 2014.

[41] Zhao, Shuai, Xiaohong Li, Guangquan Xu, Lei Zhang, and Zhiyong Feng. “Attack tree based android malware detection with hybrid analysis.” In Trust, Security and Privacy in Computing and Communications (TrustCom), 2014 IEEE 13th International Conference on, pp. 380-387. IEEE, 2014.

[42] Cui, Xingmin, Da Yu, Patrick Chan, Lucas CK Hui, Siu-Ming Yiu, and Sihan Qing. "Cochecker: Detecting capability and sensitive data leaks from component chains in android." In Australasian Conference on Information Security and Privacy, pp. 446-453. Springer, Cham, 2014.

[43] Lindorfer, Martina, Matthias Neugschwandtner, Lukas Weichselbaum, Yanick Frantantonio, Victor Van Der Veen, and Christian Platzer. "Andrubis—1,000,000 apps later: A view on current Android malware behaviors." In Building Analysis Datasets and Gathering Experience Returns for Security (BADGERS), 2014 Third International Workshop on, pp. 3-17. IEEE, 2014.

[44] Zheng, Min, Mingshen Sun, and John CS Lui. "DroidTrace: A ptrace based Android dynamic analysis system with forward execution capability." In Wireless Communications and Mobile Computing Conference (IWCMC), 2014 International, pp. 128-133. IEEE, 2014.

[45] Amini, Shahriyar. "Analyzing mobile app privacy using Computation and Crowdsourcing." PhD diss., Carnegie Mellon University, 2014.

[46] Lindorfer, Martina, Matthias Neugschwandtner, and Christian Platzer. "Marvin: Efficient and comprehensive mobile app classification through static and dynamic analysis." In Computer Software and Applications Conference (COMPSAC), 2015 IEEE 39th Annual, vol. 2, pp. 422-433. IEEE, 2015.

[47] Bhandari, Shweta, Frederic Herbreteau, Vijay Laxmi, Akka Zemmari, Partha S. Roop, and Manoj Singh Gaur. "POSTER: Detecting Inter-App Information Leakage Paths." In Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security, pp. 908-910. ACM, 2017.

[48] Clarke, Roger. "Internet privacy concerns confirm the case for intervention." Communications of the ACM 42, no. 2 (1999): 60-67.

[49] Lin, Jialiu. "Understanding and capturing people's mobile app privacy preferences." PhD diss., Carnegie Mellon University, 2013.

[50] Felt, Adrienne Porter, Serge Egelman, Matthew Finifter, Devdatta Akhawe, and David Wagner. “How to Ask for Permission.” In HotSec. 2012.

[51] Felt, Adrienne Porter, Kate Greenwood, and David Wagner. “The effectiveness of application permissions.” In Proceedings of the 2nd USENIX conference on Web application development, pp. 7-7. 2011.

[52] Fahl, Sascha, Marian Harbach, Thomas Muders, Lars Baumgartner, Bernd Freisleben, and Matthew Smith. “Why Eve and Mallory love Android: An analysis of Android SSL (in) security.” In Proceedings of the 2012 ACM conference on Computer and communications security, pp. 50-61. ACM, 2012

[53] Nagappan, Meiyappan, and Emad Shihab. “Future trends in software engineering research for mobile apps.” In Software Analysis, Evolution, and Reengineering (SANER), 2016 IEEE 23rd International Conference on, vol. 5, pp. 21-32. IEEE, 2016.