SOME SECURITY INVESTIGATIONS ON SMARTPHONE PLATFORM

A thesis submitted in partial fulfilment of the requirements

for the award of the degree of

DOCTOR OF PHILOSOPHY

in

ELECTRONICS AND COMMUNICATION ENGINEERING

by

SUMIT KUMAR

(2K17/PhD/EC/08)

under the supervision of

Prof. S.Indu & **Dr. Gurjit Singh Walia**

DEPARTMENT OF ELECTRONICS AND COMMUNICATION

DELHI TECHNOLOGICAL UNIVERISTY

DELHI-110042

May 2022

DELHI TECHNOLOGICAL UNIVERISTY

CERTIFICATE

This is to certify that the thesis entitled "**Some Security Investigations on Smartphone Platform**" being submitted by Sumit Kumar (Reg. No.: 2K17/PhD/EC/08) for the award of degree of Doctor of Philosophy to the Delhi Technological University is based on the original research work carried out by him. He has worked under our supervision and has fulfilled the requirements, which to our knowledge have reached the requisite standard for the submission of this thesis.

It is further certified that the work embodied in this thesis has neither partially nor fully submitted to any other university or institution for the award of any degree or diploma.

Prof. S. Indu
Supervisor
Professor
Dept. of ECE.
Delhi Technological University

Dr. Gurjit Singh Walia
Supervisor
Scientist 'F'
Scientific Analysis Group
DRDO

Prof. N.S. Raghava
Head of the Department
Dept. of Electronics and Communication
Delhi Technological University

Declaration of Authorship

I hereby declare that all information in the thesis entitled "**Some Security Investigations on Smartphone Platform**" has been obtained and presented in accordance with the academic rules and ethical conducts as laid out by Delhi Technological University. I also declare that, as required by these rules and conduct, I have fully cited and referenced all materials and results that are not original to this work.

(**Sumit Kumar**)
Research Scholar

Acknowledgements

First and foremost, thanks to the Almighty for giving me strength and inspiration to carry out this research work. I owe a deep sense of gratitude to all his comprehensive soul whose divine light has enlightened my path throughout the journey of my research.

I would like to express my sincere and heartfelt thanks to my research supervisor **Prof. S. Indu** for her valuable guidance, enthusiastic encouragement and persistent support. I am truly grateful from the core of my heart for her meticulous approach, wonderful assistance of her perspective and fruitful discussions on the research topic. Her careful supervision and personal attention have given me a lot of confidence and enthusiasm, during the different stages of my doctoral investigations.

I place on record my heartfelt gratitude and sincere thanks to **Dr. Gurjit Singh Walia** who has been my supervisor, advisor and mentor. He is the one whose expertise in the field is widely acclaimed. I thank him for his valuable advice, constant support and revered guidance. I invariably fall short of words to express my sincere gratitude for his patience and motivation.

I express my sincere gratitude to the **Prof. N.S. Raghava, Head of Electronics and Communication Department** for his endless support and cooperation. I am also thankful to **Prof. Neeta Pandey, DRC Chairman,** Electronics and Communication Department for constantly encouraging and supporting me in this endeavour of mine.

I lay my indebtedness to my current organisation where I am working, Scientific Analysis Group, Delhi, for exhibiting a faith in me and extending cooperation during the process of carrying out my research work along with my professional responsibilities in the organisation. I am thankful to the **Director, SAG** and all staff members of SAG, Delhi for their kind help and support during the entire period of my research.

I dedicate this thesis to my family for their endless love, support, encouragement and blessings throughout my academics. My father, **Shri Dhani Ram** not only raised and nurtured me but also taxed himself dearly over the years for my education and intellectual development. My mother, **Smt. Pushpa** has been a source of motivation and strength during moments of despair and discouragement. I am indeed grateful to my wife **Smt. Jyoti Singh** for her continuous support, care and motivation. I am also thankful to my daughters, **Shreyashi** and **Tushika** for their immense cooperation as I could not spare sufficient time for them due to my very busy research work.

Sumit Kumar

This thesis is dedicated to my parents.

For their endless love, support and encouragement

Abstract

Security investigations on smartphone platform is an interesting field of mobile security. Amongst different smartphone platforms available in the smartphone ecosystem, Android is the most widespread platform because of its open architecture. Unluckily, android based smartphones have progressively turned into the key target of the attackers, thereby enforcing urgency for security investigations. Vulnerabilities in the android smartphone platform occur due to numerous weaknesses inherent in the smartphone's software, hardware, OS, firmware, and applications. These weaknesses are exploited by the attackers to extract sensitive information by articulating a plethora of attacks. The rising popularity of apps has enticed attackers to design malicious apps (malapps) to extract critical information such as banking credentials, social networking passwords, official documents, contacts, etc.

These malapps are evolving and using novel techniques to target smartphones. These malapps are designed to evade detection and mitigation techniques. The traditional detection tools trust mostly on signature-oriented approaches and hence are not able to recognise sophisticated malapps. Thus, there is a need to design techniques for improved malapp identification and classification. There is also dearth of adequate research on scrutinising the threats posed by malapps.

The main aim of this study is to address these issues and offer powerful solutions. A lot of solutions have been proposed based on the static, dynamic, hybrid, and traffic analysis approaches. But designing and developing a robust framework by fusing the various static, dynamic, and traffic features is tiresome and demands further research. Therefore, it is indispensable to develop solutions encompassing both feature and

score level fusion that can handle the challenges in the detection of various malicious applications.

Feature fusion comprises an optimal fusion of various static, dynamic, and traffic features resulting in a unified feature. This unified feature is fed to the ensemble of parallel classifiers and their respective scores are optimally fused. The objective of this thesis is to suggest robust static, hybrid, and traffic-based frameworks to detect the vulnerabilities in smartphone platform.

To address the issues in the static analysis, a smartphone security analysis technique based on the amalgamation of multiple static features followed by fusion of scores of three classifiers connected in parallel has been proposed. The performance of the proposed static analysis technique is experimentally validated using chimeric databases.

But the static analysis approaches fail to detect run-time behaviours of malapps. To address this issue and an optimal unification of static and dynamic features for smartphone security analysis has been proposed. The proposed solution exploits both static and dynamic features for generating a highly distinct unified feature vector using graph-based methods. Further, a unified feature is subjected to the fuzzy-based classification model to distinguish benign and malicious applications. The suggested framework is extensively experimentally validated through both qualitative and quantitative analysis and results are compared with the existing solutions. Performance evaluation over benchmarked datasets revealed that the suggested solution outperforms state-of-the-art methods.

Some malicious applications are detected solely on the traffic based characteristics. There are multitude of traffic features that can be exploited for detection of malapps.

But the fusion of complementary traffic features are not exploited till date for the detection of malapps. To address the traffic analysis problem, a novel traffic feature based analysis framework wherein multiple traffic features are optimally combined to generate unified feature for the detection of unintended functionality has been proposed. Generated unified feature is then given to classifiers to get corresponding classifier scores. The score fusion method is further employed to get the final score for the detection of unintended functionality arising out of the malicious application. The robustness of the suggested framework when evaluated on the standard datasets outperforms contemporary techniques.

Thus, by developing these novel techniques, all major issues regarding the smartphone platform security analysis have been addressed. This thesis incorporates the developed techniques and their performance evaluation along with future directions.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Smartphone security analysis primarily deals with the fortification of a smartphone from threats and vulnerabilities posed by anomalies in hardware, software, firmware, OS, and applications. In other words, smartphone security analysis or investigations mainly deals with finding out the root cause of the security breach and how far it compromised or threatened the security of information in the smartphone. The static, dynamic, hybrid, and traffic analysis techniques 1are employed for the smartphone security analysis.

The static, dynamic, and hybrid analyses cater only to detecting malicious applications. Through traffic analysis, in addition to the malicious application, anomalies in the OS, firmware, and hardware are also detected. In static analysis, only static features are exploited for the identification of malicious applications. Those applications which exhibit malicious behaviour during the run time are analysed by dynamic analysis. Both static and dynamic features are exploited in hybrid analysis. It is useful for detecting malicious apps that are smart and behaves capriciously. Through traffic analysis, traffic features were collected and further exploited to detect the malwares and other anomalies in smartphones. All the static, dynamic, and traffic analysis techniques have to be further explored for the smartphone security analysis due to the inherent nature of the vulnerabilities in the smartphone ecosystem.

The main cause of security breaches in smartphones is malicious apps [1]. Hence, smartphone security analysis can broadly be categorized as apps security analysis. Many smartphone platforms allow applications to be run on them. The Android, Windows Phone, iOS, Blackberry OS, Kai OS etc. are some of the smartphone platforms.

For addressing the issues concerning various aspects of smartphone security analysis, we have conducted our research and developed several techniques.

1.1 Smartphone Security Analysis

Smartphone has now become an essential part of human life due to their myriad of benefits viz. telephony, social networking, banking, e-commerce, messaging, video teleconferencing, etc. Android OS Smartphones have captured more than 85.1% [2] of the market share. With its openness, popularity, and over-reliance, attacks on the Android platform are also engulfing. Since the Android platform permits the installation of apps from unconfirmed and third-party sources, it makes the circumstances more difficult for the users. Data kept in a smartphone (such as banking credentials, social networking passwords, official documents, contacts, etc.) provoke assailants to devise methods to obtain this critical data illicitly by employing Android malwares such as Trojans, backdoors, worms, botnets, spyware, aggressive adware and ransomware [3].

Generally, malwares are embedded in the popular android apps by repackaging techniques [4] to pass these malicious apps as benign apps and make them susceptible to malware attacks and security vulnerabilities. These malapps are produced to accomplish diverse attacks like pilfering personal info, transferring messages without permission, enticing users to malicious sites, and posing a grave risk to smartphone

operators. To elude detection, malapps are continuously evolving with many variants that further take a formidable challenge to identify. Also, the hackers are designing the malwares in a way to evade the Machine Learning (ML) classifiers [5]. As a result, effective and proficient detection methods are desperately needed to handle the growing complexity of Android malware. To tackle the numerous challenges in Android malapps detection, the research fraternity has produced voluminous work in this arena.

There are conventionally two methods for the security analysis of smartphone platforms viz. Static [6] and Dynamic [7]. Researchers have also proposed a combination of static and dynamic methods resulting in the hybrid analysis [8] to identify the existence of mobile malware.

In Static analysis, a source code walkthrough is performed to cater to software-based app-related vulnerabilities without the execution of the code. On the other hand, dynamic analysis can be performed in two different ways. One is using the execution of applications in a sandbox environment and the other way is through traffic analysis. The former method deals with software-related vulnerabilities while traffic monitoring assists in software as well as hardware-related vulnerabilities/issues such as hardware trojans etc. Also, the complexity of open source OS is very high and put a hindrance in the analysis of OS-related vulnerabilities through static analysis. This motivates the researchers to perform dynamic analysis of smartphones via traffic monitoring which is also the key focus of our survey. The classification of the techniques used in the security analysis of smartphones is illustrated in Fig 1.1.

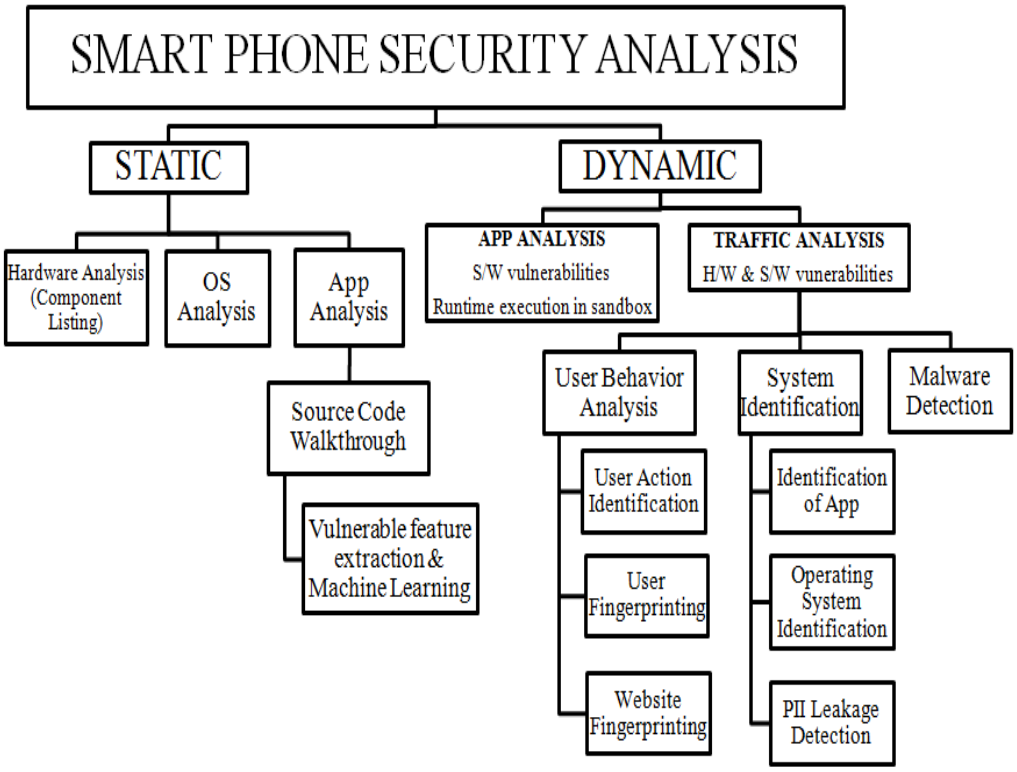

Figure 1.1: Classification of Smart Phone Platform Security Analysis Techniques

Smartphone traffic monitoring or traffic analysis is a Network traffic analysis. It is the process of recording, intercepting, reviewing, and analysing smartphone traffic to detect and respond to security threats. This branch of computer science deals with inferential methods that are responsible for converting network traces of devices into detailed statistics about their users, the apps installed on their devices, and the type of traffic/network packets or data that flows through a network. Capturing of network traces is done at different layers mainly the application layer, data link layer and at different points including within the devices or a Wi-Fi network, and contains encrypted content which makes the analysis more challenging. The traffic statistics [110] from network traffic analysis help in evaluating and understanding the download/upload speeds, origin, destination, size, type, the content of packets, and network utilization. Network utilization is basically the ratio of the amount of traffic

in the network to the maximum amount that a network can support. Network traffic analysis is also used to identify suspicious apps with the help of traffic captured.

The static solution includes scrutinizing the app without actually running it and uses techniques such as disassembly, de-compilation [3] or pattern matching, and so on. It is favourable because of being faster and inexpensive. However, it fails against the codes that use transformation, encryption, and polymorphic techniques as well as it is unable to detect new malware families and monitor the mal-app behaviour during runtime. In contrast, a dynamic solution comprises the execution of the app to take account of its run-time behaviour. It includes features of smartphones for instance CPU usage and power, and the number of processes running, to find out the existence of mal-behaviour. Malapps that download malware at runtime and escape static permission-based uncovering can be detected via network traffic. The hybrid analysis combines the advantages of both dynamic and static analysis techniques and can give in better accuracy. To detect mal-functionality in the apps, it combines run-time data extorted from the dynamic investigation into a static analysis algorithm.

As shown in Fig 1.1, the static analysis covers the hardware analysis, OS analysis, and App analysis. Static analysis of hardware is the identification of the hardware components (ROM, RAM, ICs, microcontrollers, FPGA, etc.) along with tracing circuit flow diagram with the aid of a multi-meter to understand how the various components are interconnected. It gives the idea about the basic functionality of the hardware. Static analysis of OS(s) and Application(s) is fundamentally a source code walkthrough. Static analysis of application(s) is the extraction of the vulnerable/distinguishing features and then devising a model using machine learning algorithms to detect the malwares. Under dynamic analysis, application (apps) analysis and traffic analysis is done. Dynamic analysis of apps is done by executing the

application in the sandbox environment. A sandbox is an inaccessible testing atmosphere that allows users to execute programs or open documents without disturbing the platform on which they run. Here the software vulnerabilities of the applications in the runtime environment are extracted. Traffic analysis [9] of smartphone devices also comes under dynamic analysis. Here both the software-based and hardware-based vulnerabilities can be studied by analysing the network traffic generated by the smartphones. In traffic analysis, apps are run either on a smartphone or the emulator, and corresponding traffic generated is captured and stored. From the stored traffic, traffic features are extracted to train a machine learning model to detect the malapps. The main aim of this research is to analyse the risks and devise the solution to mitigate these risks that may arise due to the daily use of the applications in smartphones.

1.2 Thesis Overview

The thesis comprises of six chapters and a brief description of these chapters is given below:

Chapter 1: This chapter covers the introduction to the topic of smartphone security analysis. It will also contain thesis overview, research problem and the objectives of the research work.

Chapter 2: This chapter covers the state-of-the-art techniques developed in existing research work on “Some Security investigations on Smartphone Platform”. It will also highlights the research gaps in the existing work that has stimulated the development of research objectives. In addition, evaluation metrics and benchmark datasets require for the performance validation of the proposed frameworks are discussed.

Chapter 3: This chapter highlights the details of the methodology adopted to accomplish the static analysis based framework. In addition, it will also cover the observations and discussion of results.

Chapter 4: This chapter highlights the details of the methodology adopted to accomplish the hybrid analysis framework. The obtained experimental results will also be elaborated against the other compared state-of-the-art.

Chapter 5: This chapter highlights the details of the methodology adopted to accomplish the traffic analysis based framework. The experimental results on benchmarked datasets were also compared with other comparable method.

Chapter 6: This chapter contains the brief summary of all the ideas, observations and contributions of the resultants obtained in each objective. Also, the future directions are sketched in this chapter.

1.3 Research Motivation

Security investigation on the smartphone platform is an imperative field of mobile security which mainly aim to detect malapps in the smartphone. A lot of work under various malapp detection framework based on static, dynamic and traffic analysis approaches has been proposed to keep track android malwares. But it is still open and challenging to detect the malapps due to code obfuscation techniques and continuous changing nature of the behavioural conditions of the ever evolving malwares in the smartphone ecosystem. To detect such malapps, a single method of smartphone security analysis is not sufficient to provide robust detection solutions. Most of the available research work is not efficient enough to detect the sophisticated malwares. Hence, development of a robust and adaptive malapp detection model is paramount that can address these challenges. This work is motivated by the fact that multiple

complementary features are necessary for developing robust solutions. The feature level fusion followed by score level fusion of parallel classifiers is another direction that can be evaluated with the aim to provide better malapp detection. Furthermore, these detection solutions were evaluated over limited datasets only. Optimization of various hyper parameters can be explored to provide better recognition performance. Most of the static analysis approaches detect malapps either on signature based approaches or without unification of features. Feature unification followed by optimal fusion of classifiers scores needs to be investigated. Hybrid analysis based detection can be explored further to detect the malapps exhibiting dynamic behaviour. Some apps shows malicious behaviour by examining their traffic features. This problem can be resolved by designing and developing solutions based on traffic characteristics to protect the smartphone against various security threats. Hence, detection of malapps having complex and sophisticated malwares becomes cumbersome with conventional approaches. Therefore, there is a requirement to develop a novel and proficient methods for detecting these types of malapps swiftly and accurately.

1.4 Research Problem

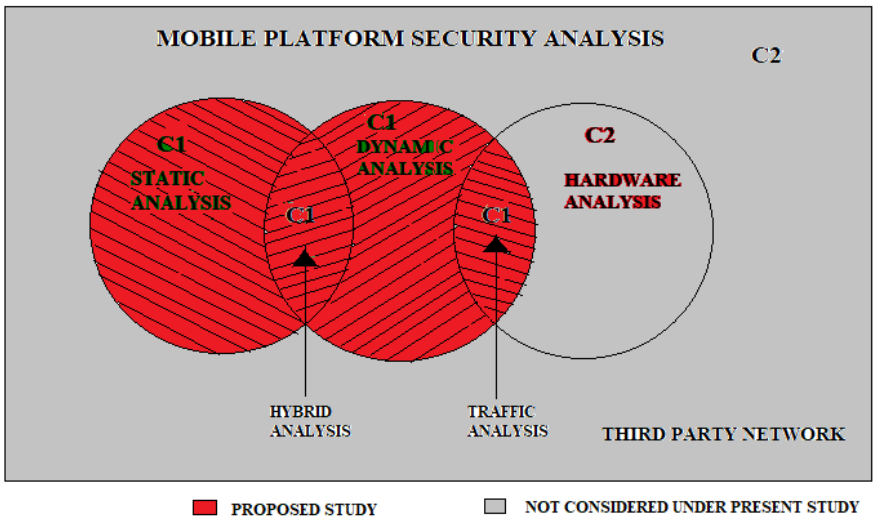

Figure 1.2: Domain of the Research Work

The Smartphone platform security analysis is vastly divided into five categories as shown in Fig.1.2 as static analysis, dynamic analysis, hybrid analysis, traffic analysis and hardware analysis. The scope of our proposed study is restricted to static, dynamic, hybrid and traffic analysis. Hardware analysis employs specialized tools and techniques to probe into hardware modules. This aspect of Security Analysis is not covered under this study. As dynamic analysis is carried out while mobile app is running, usually this approach is not only more complex but also has a high inclination towards false positive outcomes. In addition, some malware infected mobile app can easily intrude into the system just after their installation thus making the detection accuracy very low. On the other hand, Static analysis approach considers detecting malware in mobile apps by analyzing the source code segments. Various vulnerable features or resources are extracted from the application package .Vulnerable features such as permission calls, API calls, system monitor events, etc., can be easily obtained from the respective application package. Since the app is not executed, they don't require a host system environment. Hence, Static approach for malware analysis is not only computationally economical but also more accurate and efficient. Recently, Hybrid approach using both static and dynamic features was also investigated for analysis of apps. Further, machine learning based classification techniques were investigated to automate and boost the process of static malware analysis. Binary classification methods such as Random Forest, SVM, Naive Bayesian and Rotation Forest etc. provide effective ways to analyze malware patterns. In spite of most of the research investigation in literature, most of the solutions for efficient classification of apps into malicious or benign are in stage of infancy. Ever threat of malicious apps puts great hindrance to mobile users for using apps for critical applications. But, static analysis is threatened by obfuscation, polymorphism. Dynamic analysis trail the sensitive info at run-time. Dynamic analysis is superior to static analysis but needs

adequately huge set of implementations to cover app's behavior. Hence, carrying out dynamic investigations on resource constrained smartphones is challenging. To enhance the detection capability, investigators start exploiting the network traffic based solutions. Using network traffic resulted in amazing results in terms of determining concealed malwares as some malwares in malapps typically exhibits malicious characteristics only when connected in a network.

1.5 Objectives of Research Work

The objectives of this research work were to develop techniques and methods to address key questions in the study of Security Investigations on Smartphone Platform. These specific objectives are summarized as follows:

Objective 1:

- To review of the existing literature and comparison of the merit and demerit for smartphone-based platform security.
- To create of data base with large set of benign and malign apps and incorporating zero day attack.

Objective 2:

- To design and develop Static Analysis techniques for smartphone platform using multiple feature unification and optimal classification.
- To do performance comparison with existing methods on benchmarked dataset containing large set of benign and malicious apps.

Objective 3:

- To design and develop Hybrid Analysis techniques for smartphone platform through optimal combination of static and dynamic features.
- To identify feature sets and segregation of feature into different threat

level for platform security. Optimal tuning of classifier considering threat level of features.

- To do performance comparison of proposed framework with existing methods using benchmarked dataset.

Objective 4:

- To design and Develop traffic analysis framework for optimal combination of multiple features for detection of unintended functionality.
- To do performance comparison with existing methods using online captured data from ten different apps under unconstrained environment.

1.6 Thesis Contribution

In this thesis, we show that the key to effective malapps detection relies unification of extracted features followed by multiple classifiers, which eloquently delineates the performance capabilities of a framework. Through this set of static, dynamic and traffic features three independent frameworks viz. static analysis, hybrid analysis and traffic analysis have been designed and developed.

The principal contributions of this thesis are:

(i) We first introduce a realistic static feature approach for smartphone security analysis that incorporates multiple feature unification through cross iterative diffusion. To our awareness, it is the first time that this approach is introduced to extract unified android static features. Pragmatic and effective app security analysis framework is proposed wherein three ML algorithms are exploited to evolve a system to detect the malapps on the basis of unified feature representation. Further, outcomes of the ML algorithms were fused by

DSmT[153] algorithm to improve the accuracy achieved by individual classifiers. In addition, we presented a complete investigational study based on CICMalDroid2020[132], AMD[129] and Drebin[128] malapps database and comparative experimentations with state-of-the-art methods to validate the efficiency and proficiency of our approach.

(ii) To detect the apps exhibiting the dynamic behaviour, we put forward a unique approach for optimal unification of static and dynamic features resulting in Unified feature (UF) for smartphone security analysis by cross diffusion technique. Then this UF is fed to two ML classifiers to detect the android malapps. Results of these classifier's scores were combined by fuzzy based fusion approach for improving the performance. Lastly, we provided a comprehensive study founded on benchmarked databases and compare the results with contemporary techniques to validate the efficacy of the suggested framework

(iii) To design the traffic based analysis framework, we proposed a traffic feature-based fusion that comprises of optimal combination of multiple traffic features by cross-diffusion of order and sparse graphs to produce a unified feature. The unified feature vector thus generated is given to the three parallel ML classifiers and classifiers scores obtained are fused to enhance the accuracy attained by separate classifiers. Presented the performance comparison with existing state-of-the-art methods using standard data sets available.

Chapter 2

Literature Review

In the last few years, smartphone security analysis has been widely investigated and reviewed. Specifically, the success of any smartphone security analysis technique is greatly reliant on developing an efficient and effective model. In this direction, the smartphone security analysis approaches have been suggested under static analysis framework, dynamic analysis framework hybrid analysis framework and traffic analysis framework. The various smartphone security analysis models are concisely reviewed. The most popular smartphone android platform, malwares in smartphone ecosystem, features exploited for detection of malwares in smartphone applications, fusion techniques, performance metrics, benchmarked datasets, and research gaps along with smartphone security analysis models were also reviewed.

2.1 Smartphone Platform

Smartphone platform basically comprises of hardware, an OS and software/drivers for a particular microprocessor. Platform host numerous applications (apps) and allow these apps to execute on them.

In this section, the overview of Android, its salient features, the architecture of Android OS, various malware types and the ways by which malware apps infects the end users are discussed.

2.1.1 Android Overview

Android is open source and Linux-based OS designed for smart devices i.e. mobile phones and tabs. Android's latest version i.e. 12L (API level 32) [10] was released on 7 March 2022. Google and Open Handset Alliance (OHA) developed Android which was launched in 2007. The Android smart devices contain lots of features and functionalities which includes hardware features such as audio, Bluetooth, camera, network, microphone, and sensors such as accelerometer, barometer, compass, gyroscope, and Wi-Fi. It also includes software features such as app widgets, live wallpapers, storage, messaging, multi-language support, browsers, media support, call, messaging, multitasking, external storage and so on. Android is best-selling OS in the world for smartphones since 2011 and for tablets since 2013. More than 3.48M apps were available on Google Play Store for download, as of Aug 2021. Android Apps are developed via SDK and using mainly JAVA programming language. For the development of shared library and native code, C and C++ languages are used. The support for development of app in Kotlin programming language was announced by Google in May 2017. Google Play Store is the official app store comes inbuilt in Android. Google Play Store allow users to browse, download and update applications from the house of Google.

2.1.1.1Android Architecture

Android software stack has mainly four layers and is divided into five sections. The core architecture of the Android is depicted in Fig.2.1:

(i) Linux Kernel

It is the heart of android OS and is at the bottom of the android architecture. It forms as an abstract layer b/w the hardware devices & rest of the layers of the software stack. It provides services such as power management, resource access, device management, memory management and device drivers for hardware like the device display, camera, Wi-Fi, keypad and audio.

(ii) Libraries and Android Runtime (ART)

The pre-defined libraries include C/C++ core and Java based libraries like OpenGL, WebKit, FreeType, Media, SQLite. The SQLite library is responsible for database, FreeType handles font support, WebKit is accountable for browser support and SSL manages Internet security. Android Runtime environment includes components like core libraries and the DVM i.e. Dalvik virtual machine. DVM is just similar to JVM(Java Virtual Machine) but it has been optimized for mobiles. It utilizes multi-threading and memory management, the Linux core features, which are crucial in JAVA. ART includes features such as Ahead-of-time (AOT) and just-in-time (JIT) compilation, better debugging support, optimized garbage collection (GC).

(iii) Application Framework

This layer provides several services to apps in the form of interfaces and Java classes. App developers exploit these higher-level services in their apps. Besides, it provides an abstraction for h/w access and manages the UI and app resources. Application framework includes Android API's such as locations, package managers, resources, UI, Content Providers (data) and telephony.

(iv) Applications

It contains native apps and apps by third party developers such as browser, mail, gallery, SMS, contacts and clock that are created using the Application framework. For example, SMS app is used to deliver a message to the user specified recipient.

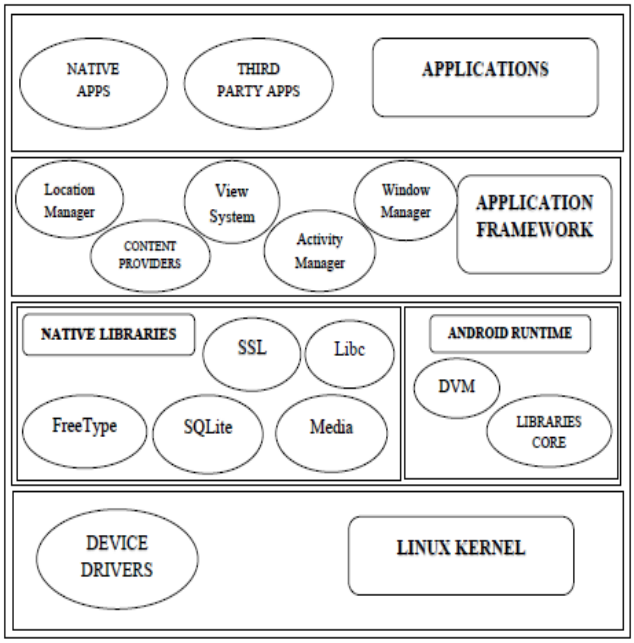

Figure 2.1. Android Operating System Architecture[1]

2.2 Smartphone Platform Malwares and vulnerabilities

Malware is "intrusive software", specifically designed to gain unauthorized access to a system. These malware programs are responsible for deleting, stealing or encrypting sensitive data, monitoring users' activity without user's consent and altering core functions. Different types of malware contain unique traits and characteristics which are as follows:

[1] Source: https://developer.android.com/guide/platform

Trojan horse is a destructive program that gains access to system by looking similar to an authorized program. Trojans allow personal info theft through a backdoor entry of malicious programs to the system. Worm spread by exploiting vulnerabilities in OS and hence does not require directives of malware designers. It has the ability to make copies of itself. Also, it offers no harm to the data or file on the system. The primary characteristic of a Virus is that it can execute and replicate itself. It spreads through infecting files or programs on the system. Virus is risky in comparison to computer worm as it has the ability to delete or change the files whereas a computer worm can just replicate itself. Viruses includes Macro viruses, File viruses, Stealth viruses, Polymorphic viruses [3]. Spyware may or may not require permission to get installed on users. Its purpose is to gather data, browsing history, observe the user activity and transfer to remote user. Spyware gets installed unknowingly on installing other freeware's. It is similar to adware. It has the ability of downloading malware programs and installing on the system. Keyloggers also known as system monitors, keeps an eye on user activity that includes internet surfing, keystrokes and emails. Ransomware's [3] purpose is infecting user device and encrypting the data in order to demand a money from the user to decrypt the system data.

In smartphone security landscape, there are basically five vulnerability vectors viz. app, device, network interfaces, device content, and hardware related vulnerabilities.

2.2.1 App vulnerabilities

Due to their shorter software development cycle, smartphone app developers resorted to the open source libraries, tools and packages to develop new apps. It resulted in numerous vulnerabilities [11] due to these developed apps. These apps are prone to security flaws inherent in the open source system and put confidential data in the smartphones at higher risk. Many people download the apps from the

third party resources which are not for reviewed. These apps further makes the confidential data vulnerable to attackers.

2.2.2 Device vulnerabilities

Device vulnerabilities [12] comprises of smartphone OS and firmware vulnerabilities. These vulnerabilities are mitigated by the patches released from time to time by the OS developer. Smartphones having relatively old OS and firmware may not get support from the OS developer regarding patches and are more prone to attacks. Regular updates from the device manufacturer releases patches and ensures mitigation of these vulnerabilities. It is duty of the user to incorporate these patches to reduce the vulnerability window. Adoption of the latest OS further reduces the vulnerability.

2.2.3 Networks vulnerabilities

Vulnerabilities related to smartphone n/w [13]are centred on exploitable hardware and software flaws in the n/w interfaces. POODLE and Heartbleed are two very famous n/w vulnerabilities. The POODLE exerts the browser to more vulnerable version by relegating the browser to a relatively low strength encryption scheme. Heartbleed exploited SSL vulnerability that let an attacker to steal 64K of data from the active memory of affected systems in one cycle.

2.2.4 Web and content vulnerabilities

Attacker's uses malicious photos, web pages, videos etc. to exploit OS or an app to gain unauthorized access to smartphone. For example, Stagefright is a s/w vulnerability in android OS related to mp4 files [14]. Mp4 files can be used to gain access to multimedia messages and download folder of smartphone.

2.2.5 Hardware vulnerabilities

Researchers exploits the Graphics Processing Unit (GPU) [15] in smartphones to snoop user's credentials by logging the keyboard. All the credentials typed were compromised. GPU processes the animations corresponding to keys on keyboard. Attackers correctly infers which key is pressed on the key board with high detection accuracy.

2.3 Issues and Challenges in Smartphone Security Analysis

2.3.1 Malware Survival Techniques

The end user device is infected through Malware applications in the following ways: Repackaging, Code Transformation, Update attack, Drive-by-Download [3]. Mostly Repackaging technique [16] is used by malware designers. This technique includes following steps: application download, app disassembling, inserting malware payload to app, reassembling app and then uploading or distribute via less monitored app-stores. Malware designers change the signature of reassembled app to prevent getting detected.. The second technique is Code Transformation, which is responsible for generating unobserved malware signatures. It includes methods such as Renaming of Package, Method or Class, Resource Encryption, String Encryption, Class Encryption, Reordering of Opcode, Insertion of Junk Code. It can be used to thwart the disassembly tools. In Update attack, malware designers use certain techniques to inject malware at runtime by attaching inflicted code to the updated version of the application. This technique is undetectable by static anti-malware methods. The other technique used by malware designers is Drive-by Download. This technique uses in-

app advertisements, clicking malicious URL, or malicious QR-code which redirects users to a webpage and incites them to download malware automatically.

2.3.2 Code Obfuscation Techniques

Code obfuscation is the camouflage of the intended meaning of the code by making the code puzzling, purposefully equivocal and more problematic to comprehend. In this subsection numerous code obfuscation techniques have been discussed that are used to hinder the static analysis.

2.3.2.1 No-operation Code Inclusion and Opcode Repositioning: No-operation code inclusion is a famous practice that alter the exe file size and dodges the signature database of anti-malware software. It also saves the apps semantics. But, it alters opcode order to change the malapp signature. Repositioning of Opcode is accomplished with "Go to" instructions to change the control-flow while protecting the semantics. These approaches are used to elude the opcode and signature oriented identification solutions.

2.3.2.2 Using Reflection APIs: Sensitive Android API within the malicious applications are searched and exploited for its detection during static analysis. Apps allows Java reflection to modify the run time behaviour of class using programmatic class instances by means of the literal strings. Data flow analysis was employed to detect the method names/ exact class. But to thwart so that automatic search for reflection API the literal strings were encrypted.

2.3.2.3 Class or Package Renaming: App in Android ecosystem is recognised by its exclusive package name. Dalvik-bytecode protects the class and package names. Numerous anti-malware exploits the name of a class or package of well-known

malwares as signature for malapp identification. Such renaming is used to elude the signature based identification.

2.3.2.4 Modifying Control-flow: Few anti-malwares software's exploits control flow investigations to identify the malwares variants. Control-flow of a program can be altered using "Go to" instructions or by adding no operation code. Though insignificant, such methods elude the anti-malware software's.

2.3.2.5 String Encryption: Strings like comments, shell-commands and Uniform resource locators disclose many things about the app. To avert such static investigations, the strings must be encrypted.

2.3.2.6 Class Encryption: Sensitive information like Digital Right Management which is used by certain anti malwares can be concealed by encrypting the complete classes.

2.4 Features used in Smartphone Security Analysis

Static and dynamic features [17]are exploited in smartphone security analysis. Static features mostly includes permissions, app components, filtered intents, API, Network address, operation code, hardware components, control flow graph, static taint analysis, file property, native code are some of the static features. Dynamic features mostly used in the literature comprises of system calls, network features, system components, battery feature, phone event, SMS event, user interaction, file operation, broadcast receiver, system command, API, dynamic taint analysis etc. Traffic features are subset of dynamic features as traffic analysis is a form of dynamic analysis. In hybrid analysis, blend of static and dynamic features are used for the identification of malicious apps. In majority of the cases TCP based traffic features were exploited in determining the application, OS and hardware related vulnerabilities were determined. In few frameworks HTTP based feature were also exploited for determining the malicious apps.

2.5 Smartphone Security Analysis Techniques

Smartphone security analysis comprises of static analysis, dynamic analysis, hybrid analysis and traffic analysis. The main purpose of the security analysis is to determine the vulnerabilities related to apps, software, hardware, OS, etc. in the smartphone ecosystem. The vulnerabilities related to apps and software are catered exclusively by the static analysis, dynamic analysis and hybrid analysis. The OS and hardware related vulnerabilities were detected by traffic analysis. Apps that exhibits malicious behaviour during run time are detected by dynamic analysis methods. Some of the apps have malwares embedded in them through repackaging techniques, these malwares can be determined by modelling the detection mechanism based on signatures if the obfuscation methods were not employed. The static analysis techniques were efficient in determining these apps. Hybrid analysis is amalgamation of both static and dynamic analysis. Traffic analysis techniques were used to determine the hardware and OS related vulnerabilities along with determining the malapps in the smartphones. The details of static, dynamic, hybrid and traffic analysis follows in the coming subsection.

2.5.1 Static Analysis

Static analysis encompasses studying the app source code by just disassembly, decompilation and without executing it. Specifically, it implies analysing APK file. This analysis consumes less resources and is fast. It is prone to obfuscation techniques and also to dynamically loaded code. It fails in case of encryption, code transformation and polymorphic malware. DAPASA, an approach for static malware detection [18] used sensitive subgraph analysis to detect piggybacked apps. In this, authors focused on distinguishable invocation patterns between the riders (malicious payloads) and the

carrier (host app) of sensitive API's for detection. DAPASA exhibit impactful performance by achieving TPR of 95 per cent and a FPR (False Positive Rate) of 0.7 per cent. The combination of 5 features with the invocation structure, it supplements approaches based on API and permission. The authors [19] outlined the machine learning Random Forest based system which monitors system events, permissions, sensitive API's and use them as a feature set from APK files. It can be applied to all apps as the four group of features are always available and can be easily obtained. Additionally, it acquires key features by static analysis approach and without dynamic tracing. The results using 10-fold cross validation achieved accuracy of 89.81%. Mlifdect [20], a framework proposed by Wang et al. overcame the limitation of detection posed by a single classification algorithm. This static method is based on multiple algorithms in which 65,804 features are extracted from multilevel features like API calls, deployment of components from android apps and permissions. This includes classification model construction by Parallel machine learning and a technique on Probability analysis called Information fusion and Dempter Shafer Theory used to generate a final classification result. This framework showed 99.7 per cent accurate detection and achieved recall of 99.8% with 0.1% FP rate (False Positive rate). FP rate is the probability that a true positive will be missed by the test. Cen et al. proposed probabilistic discriminative model [21] based on RLR (Regularized Logistic Regression). Regularised Logistic Regression is a ML algorithm that caters the overfitting problem and is used to predict the probability of a categorical dependent variable. This approach produces probabilistic output with highly precise results, uses API calls as a feature and also traversed problems in feature granularity, selection, representation and regularization. It combines app permissions and decompiled source code analysis to achieve better results. To address the gap of malware detection problem from API perspective, an automated malware detection model, Malpat [22]

is proposed. It tackles the problem of manual configuration of lists of features. Moreover, the research work is based on mining hidden patterns of malware from real world android apps and extracting highly sensitive API's. The experiments were conducted over 31185 benign apps and 15336 malicious samples. MalPat achieved high F1-score of 98.24% and recall rate of 0.9963. The authors in [23] build API sequence data sets of android apps using static analysis and uses LSTM i.e. Long Short-Term Memory algorithm to develop a 2-class classification model. It is a combination method based on CFG's (Control Flow Graphs) and ML algorithm. CFG is the graphical depiction of control flow during the execution of applications. 10010 benign apps and 10683 malicious apps were taken as a sample to conduct experiment and demonstrated that the model is 98.98 per cent precise in detection.

SIGPID [24] is based on Permission usage analysis, a 3-level data pruning approach which extracts most significant permissions that can effectively distinguish between malware and goodware. It uses machine learning based classification methods and found that only 22 permissions are significant. SIGPID is capable of detecting 93.62 per cent of malware effectively and also detects 91.4 per cent of the unknown malware. Furthermore, by this approach permission to be analysed are reduced maintaining accuracy and efficiency. In [25], the authors particularly focused on two ML assisted approaches viz. clustering & classification based on analysis of the source code and permission of apps. The root access is required in the permission-based approach and it achieves better accuracy. The source code approach uses "bag of words" model used in NLP and is only static approach that uses ML to scrutinize the whole code of an app and also involves analysis of decompiled code. Former achieved the F-Measure of 89 per cent and latter 95.1 per cent. It supplements existing signature-based anti-malware methods as they only detect when proper signatures are released. Kang et al. [26] used static analysis that included creator's info and proposed detection and

classification system for android malware. This system can analyse malicious behaviour and permissions in order to gain more accuracy. Similarity scoring algorithm is executed to classify malware into similar groups and enables faster detection. It shows 98 per cent accurate detection and is 90 per cent accurate in classification performance. CASANDRA [27] a framework based on online learning which addresses population drift, handles voluminous data and detects data streaming in at real time. In this, a novel task specific kernel (i.e. CWLK) with online learning are combined for detection. It achieved F-measure of 99.23 per cent. Scalability, flexibility and high performance are key advantages of this framework.

DroidEnsemble [28] leverages by using both string features (permissions, code patterns, API usage) and structural features (data flow graph and FCG) to effectively categorize the static behaviours of applications. SV, KNN, RF are used as three ML algorithms to evaluate performance with 1386 goodware and 1296 mal apps. The results depicted that string features are more efficacious. The structural features can be used as complementary features to compensate for insufficiency of string features. Overall, the accuracy achieved by group of both excel to 98.4%. In the work of [29] , authors presented a framework that detects android malware using multimodal deep learning. This method uses various features to reflect various attributes of apps. For feature extraction, similarity and existence-based methods have been used. The approach leverage by dealing with multiple feature and was evaluated over 41260 samples to improve detection accuracy. FalDroid [30] is an automatic approach designed to classify android malware and select malware samples according to their free graphs (frequent subgraphs). 8407 malware samples were taken from 36 families and this method was applied achieving 94.2 per cent accuracy. A multilevel DroidFusion [31], model is a classifier fusion approach that enables efficient

combination of machine learning algorithm. The results are effective against both ensemble and non-ensemble base classifiers.

Significant pairings of the permissions [32] leading to a malicious application detection were identified and extracted from apps. It can lead to detection of malapps with approximately 95.44% accuracy. For this, datasets were analysed intricately and edge weights are allocated to pairs of permission depending on their frequency of occurrence in the datasets.

An ensemble [23] of three detection models based API frequency, API calls and API sequence was created to achieve detection accuracy of 98.98%. An automated malware app detection tool7 with unique ensemble learning method using permissions and API calls with Naïve Bayes, Decision Tree (DT), Random Tree (RT) etc. to detect malwares was reported with the detection rate of 99% (approx.) with very low false positives. Authors10 generated a feature vector that represents malware features having same attributes with benign applications. In this, model learning approaches and automatic upgrade system for malapps detection using a multimodal deep learning method was proposed with accuracy of 98%. Wang et al. [33] used multiple features and ensemble of classifiers viz. KNN, SVM, NB, CART and RF for android based malapps detection through majority vote fusion method. A four layered static detection model using MD5, malevolent permissions, dangerous permissions and intents was proposed by Song et. al. [34]. Authors [6] used API calls abstraction method to decrease the number of API calls be used as a feature and three ML algorithms (KNN, RF and SVM) were used for achieving detection with 98% accuracy. A technique [35] for identifying android based malapps to automatically detects malware by extracting the multiple static features such as permissions, API calls, network addresses, and mapped these features into a single feature space vector. Further, Linear SVM and DT algorithms

were employed to implement the multi-label classification. Detection accuracy of 98% and 63% was achieved for small size families' malware and zero day malwares respectively. Oluwafemi Olukoya [36] et al. investigates a malware detection model based on sensitive permissions and UI based app descriptions. Investigational outcomes establish precision of 90%. DroidDomTree [37] excavates the tree structure of API calls in Android apps for identifying malapps. A tree structure of API calls accentuates a path flow and recognizes the layout of APIs and hence stresses the prominence of some APIs in an app. It accomplished detection rates varying from 98.1% to 99.3% using eight different classifiers (J48, AD Tree, RF, RT, AdaBoost, Naïve Bayes, Radial Basis Function Network, K Star). Although, most of the work exploited either multiple classifiers or multiple features for development of solution for smartphone security analysis, a comprehensive solution exploiting both optimal combination of classifier and efficient fusion of multiple features was not investigated. In another dimension, model based android malware detection approach were also investigated. For instance, Roni Mateless [38] et al. presented a model for malapp detection with 97.8% detection accuracy. Decompiled source code contains API calls, keywords, function names, strings in human format etc. Malevolent codes vary from the text because of the syntax rules of compilers and to prevent detection. NLP method was adapted here to classify the apps. Ke Tian [16]et al. investigates a method to detect the repackaged apps by code heterogeneity analysis. Code structure was divided into various subset and each subset was classified based on the features. Each subset depicts dependence based region. In this partition based detection, False Positive (FP) rate of 2.97% and False Negative (FN) rate of 0.35%were obtained. In MAMADROID [6], a Markov chain based behavioural model for detecting the android malware. Sequences of API calls were modelled as Markov Chain. Model has achieved the F-measure of 0.87. Qian Han28 et al. proposed malicious app detection scheme by using

irretrievable feature transformations so that the evading of ML classifiers by hackers becomes impossible.

Ensemble Rotational Forest based model [39] was proposed that exploits permissions, permission rate, sensitive API's etc. as key features to detect malware with accuracy of 88.26%. In MalResLSTM [40] , authors presented Long Short Term Memory(LSTM) based method to classify malapps. Feature extracted were mapped to vector space and processed in the LSTM based deep learning model to achieve the accuracy of 99.32%. DroidDomTree24 excavates the tree structure of API calls in Android apps for identifying malapps. A tree structure of API calls accentuates a path flow and recognizes the layout of APIs and hence stresses the prominence of some APIs in an app. It accomplished detection rates varying from 98.1% to 99.3% using eight different classifiers (J48, AD Tree, RF, RT, AdaBoost, Naïve Bayes, Radial Basis Function Network, KStar). Although, most of the work exploited either multiple classifiers or multiple features for development of solution for smartphone security analysis, a comprehensive solution exploiting both optimal combination of classifier and efficient fusion of multiple features was not investigated.

In sum, survey of the closely linked literature revealed that most of the above approaches used traditional classifiers which can detect malapps using one or more classifiers. Multiple classifiers using multiple features gives improved overall performance in comparison to the single classifier using multiple features. Also, most of work either focussed on optimal combination of features or optimal combination of classifiers. Hence, future direction of smartphone security analysis is to take benefit of both feature-level and score-level fusion.

Table 2.1 Comparison of Static Analysis Techniques.

Author's Name	Feature Set	Methodology	Dataset	Remarks
Fan et al. [18]	Sensitivity coefficient, total sensitive distance, tnsm1 (the total number of sensitive motif instances), tnsm2, tnsm3	Based on invocation patterns of sensitive API's RF, C4.5, KNN, PART	M- Android Malware Genome Project[117], VirusShare [10] B- Google Play, Anzhi Market	Complements permission based and API based approaches.
Zhu et al. [19]	sensitive API, permissions and its rate, monitoring system events	RF based Malware detection	Hemdds Dataset (B - official Android market & M – VirusShare [10])	Cost-effective alternative. Useful in user information security area.
Narayanan et al. [27]	*Semantic features:* subgraphs from CADGs viz. contextual API dependency graphs	Based on online learning Confidence Weighted algorithm as classifier	Drebin, Google Play, Anzhi, AppChina, SlideMe, HiApk, FDroid, Angeeks	Handles population drift, voluminous data, real time streaming.
X. Wang et al. [20]	android manifest.xml and Disassemble code of apps Feature set: APIC and CHPN	Uses multiple concepts from information fusion, ML, static analysis.	M- Drebin [52], Android Malware Genome Project [117] B- Google Play	Run time efficiency, Low overhead, Lightweight
Xu et al. [41]	ICC related features	Involves comprehensive analysis of ICC patterns of malware and goodware.	B- Google Play Store M- Drebin[52]	Useful for detecting "advanced malware"
Cen et al. [21]	Android API calls	Based on Regularized	C11 [42], C12 [42], CM [43], O [44], OM[43]	Integrate analysis of both decompiled source code as well as app permissions.

		Logistic Regression. Probabilistic discriminative model		
Kang et al. [26]	Serial no. info from certificates	Uses Creators information Based on similarity scoring algorithm	M- VirusShare[45], Contagio Mobile and Malware.lu B- Android market, GooglePlay	Enables fast detection
Wang et al. [28]	*String features* viz. requested and used permissions, restricted API calls, code patterns, filter intents *Structural features* viz. FCG	Uses both String and Structural features. SVM, KNN, RF	*B*- GooglePlay, AnZhi[46], LenovoMM[47], Wandoujia[48] *M*- FakeInst, Opfake, FakeInstaller, DroidKungFu, GinMaster, Plankton	Reduced Android software failures.
Tao et al. [22]	Permission related API's	Based on mining hidden malware patterns RF classifier	B- BenignAll (consists of two datasets BenignRan & BenignPop) M- VirusShare Contagio	Surpassed approaches such as MUDFLOW,DREBIN, DroidAPIMiner
Ma et al. [23]	API info from CFG	*Detection model* viz. API frequency, API sequence and API usage C4.5, DNN, LSTM algo	Benign apps from AndroZoo & malwares from AMD	Reduced analysis time. Detect unknown malware

Kim et al. [29]	String, opcode, method API, component, method opcode, permission, and environmental features.	Multimodal deep neural network model	*Malware samples:* VirusShare, Malgenome project *Benign:* Google Play app store	First application of multimodal deep learning. Detect even when malware has many features similar to benign apps
Fan et al. [30]	Frequent subgraphs, a novel graph-based feature selected from FCG's	Representative malware selection Builds Frequent subgraphs SVM, KNN, C4.5, RF	FalDroid-I, FalDroid-II, Drebin and Android Malware Genome Project dataset	Concept drift problem. Unreliable against native code.
Li et al. [24]	Significant permissions	Based on permission usage analysis Multi-level data pruning approach	Google Playstore and Anzhi Store [46]	Efficient in detecting new/unknown malware samples. Effective when compared to existing virus scanners.
Yerima et al. [31]	Permissions, intents, API calls, possible external payloads (e.g. hidden .dex files)	Involves Classifier fusion approach Based on multilevel architecture	Malgenome-215, Drebin-215, McAfee-350 and McAfee-100	Outperformed stack generalization. Not able to handle multiclass problems.

2.5.2 Dynamic Analysis

In dynamic analysis, an app is executed in a sandboxed (protected) environment, and its behaviour is monitored. Unknown malware can be detected through this analysis and it is effective against malware obfuscation. But it consumes more computational power and takes a longer time than static analysis.

Cai et al. [49] developed a dynamic app classification technique called DroidCat, using variety of features such as ICC(Inter Components Communication) intents and method calls. ICC intents provide a mechanism for data exchanging between

components. It proved powerful against system call, resource obfuscation, reflection, and run-time permissions. DroidCat attain 97 per cent F1-Measure accuracy for detecting or categorizing malware. MADAM [50] , a multilevel malware detector which detects misbehaviours of almost all apps by analysing behavioural characteristics. It extracts 5 set of features from 4 levels i.e. kernel, package, application & user level. MADAM identifies security risk by analysing permissions and meta data like download number and user scores at installation time, and if detected risky, the app is marked as 'suspicious'. Detection was done over 3 large datasets having 2800 apps and effectively block more than 96% of mal apps.

Endroid [51], an approach blends dynamic analysis by employing ensemble learning, identifies malware family and detects android malware. Dynamic behavior such as app-level malicious and system level behavior are automatically extracted. It removes irrelevant features by adopting feature selection algorithm and selects critical features responsible for risky behaviors.

CANDYMAN [52], a tool for family classification that combines deep learning (Markov Chains) and dynamic analysis. In this dynamic analysis, Markov chain models the probability between the sequence state acting as feature in classification and information of malware samples is excerpted as sequence of states. Using feature space classical machine learning algorithm was trained that included deep learning and imbalanced learning methods. The performance of 81.8% was achieved using dataset of 4442 mal samples of 24 different families.

In sum, dynamic analysis uses features as system calls, system components and user interaction and is useful for segregating those vulnerabilities that exhibits maliciousness during runtime.

Table 2.2. Comparison of Dynamic Analysis Techniques.

Author's Name	Feature Set	Methodology	Dataset	Remark
Cai et al. [49]	ICC intents, structure and security relevant API's	App level profiling Supervised Learning (RF)	AndroZoo , Google Play, VirusShare [45], Drebin dataset , Genome	Robust against reflection and evasion techniques like system call obfuscation
Bhatia et al. [53]	Frequency of System call	J48 Decision Tree and RF algorithm. System Call-based detection	Android Malware Genome Project (*mal apps*), Google Play store (*benign apps*)	Effective in classifying unbeknownst app & detecting and monitoring behavior of apps that employ complex obfuscation technique.
Singh et al. [54]	System Call behavior	*Data classification Technique:* Decision Tree, RF, Gradient boosting trees, KNN, ANN, SVM and deep learning.	Contagio Project (Parkour, 2016) *mal apps & normal apps* from Google Playstore.	Found 31 sys calls from total of 337 as extremely good predictors of malware apps.
Zhang et al. [55]	Markov chains & Determinate System calls	Contribution level-based method Markov chain model construction SVM classifier	*Benign apps & malware apps* provided by Zhou et al. [43]	Reduced number of distinct sys calls to speed up the identification process.
Yalew et al. [56]	Uses Kernel system and API function calls	Leverages the Trust Zone Extension KNN classifier	*Malware samples* (2014-16) from the Contagio mobile repository & *Benign* from Google playstore	Provides second layer of protection at run-time.
Iqbal et al. [57]	CPU, memory usage, Linux Kernel System calls	4 subdetectors RF classifier	AndroZoo dataset	End users are free to use any type and install any no. of subdetectors as per requirement.
Jaiswal et al. [58]	System calls	Involves System call analysis	*Normal game apps* from google playstore &	It observed permissions requested by

		Behavior categorization: On basis of similarities & dissimilarities	*Malware game apps* from the Virus total	malware clone apps that lead to malicious behaviour.
Feng et al. [51]	Sys level trace, common app-level mal behaviors	Utilizes multiple set of dynamic features Uses Ensemble learning algorithm Adopts stacking	*Benign apps:* Google playstore and AndroZoo dataset *Malware apps:* Drebin dataset	Malware behavior may not be triggered by MonkeyRunner (dynamic analysis paltform) + DroidBox
Martin et al. [52]	Incoming & outgoing n/w data, SMS sent, cryptographic operations circumvented permission and phone calls etc.	Combined Dynamic analysis and Markov chains Utilized ML algorithms integrated with distinct imbalanced learning & deep learning techniques.	Drebin dataset	Integrates dynamic analysis and deep learning approach for classification of malware family

2.5.3 Hybrid Analysis

Hybrid analysis involves combining static and dynamic analysis features that encompasses scrutinizing code and behavior of an app. It combines advantages of both the approaches and addresses issues akin to static analysis like inability to detect obfuscated, zero-day malware and dynamic code loading, it also deals with dynamic analysis issues such as inability to examine of all execution paths of an app. The main limitation of hybrid analysis is that it consumes more Android system resources and takes a long time to perform the analysis.

Rehman et al. [59] presented a hybrid framework to overcome the limitation of signature-based methods that are not able to effectively detect polymorphic viruses and zero-day attacks. It examines both heuristic and signature-based analysis that can be used on diverse environments. In this approach two types of features are extracted from various android apps and are classified into 3 categories: keywords, user(s)

permissions from manifest.xml files and strings from other files of apps. Result showed that SVM in case of binaries and KNN (K Nearest Neighbour) in case of manifest.xml files were best suitable. In [41], authors focused on a framework based on deep belief network and combined static, dynamic analysis and system calls to extract hybrid set of features to distinguish malware app from legitimate apps.

The proposed framework achieved 99.1% detection accuracy with the presented dataset. The evaluation compares various ML approaches with deep belief networks algorithm. Moreover, author developed complete static analysis jar which adopts different effective methods in an attempt to ease and speed up the static analysis by handling all the android apps in only one step instead of considering one app at a time and it is also capable enough to check the similarity between two versions of the same app downloaded from different markets.

SAMADROID [60] is based on benefits and limitations of existing antimalware techniques. It is a 3-level hybrid system for android. It is developed by combining the benefits of (i) Static and Dynamic analysis (ii) Machine Learning Intelligence (iii) Local and remote host. It performs dynamic analysis on the device and communicates with the server for static analysis and detection results. The approach ensures low resource consumption, high detection accuracy and efficiency in terms of power and storage consumption.

The authors in [61] presented a hybrid model DAMBA based on client/server architecture. It can detect the android malapps by constructing the directed graphs depicting the object reference information with 96.9% detection accuracy and a detection time of approximately 5 seconds. Hybrid analysis approach “mad4a” [62] offered by the authors for android malapp detection leveraging advantages of both the dynamic and static analysis methods. In this, authors pointed out some undervalued characteristics of android malapp that can further help the investigators to augment

their knowledge for detecting android malapps. In this, API calls and network logs were used as static and dynamic features respectively. Manzanares et al. offered a novel hybrid analysis method “KronoDroid” [63]that addresses the time and data platform source. Here, 489 static and dynamic features were used for generating hybrid dataset with labelled timestamps on each sample to detect the malapps with high accuracy. A hybrid method “DirectDroid” [64], that merges fuzzing and a novel app analysis procedure “on-demand forced execution” to activate concealed malicious behaviour. It can effectively detect malapps by the “augmenting fuzzing” technique. This method could not cater to obfuscated codes in the malapps. In sum, hybrid security analysis techniques without ML are limitedly applied for malapp detection.

A multi-level hybrid model SAMADroid [65] for effective android based malapp detection was proposed. Static analysis results were determined on the remote host. Smartphone was used for the dynamic analysis. Machine learning algorithms Support Vector Machine (SVM), Random Forest (RF), Naïve Bayes(NB), and Decision Trees (DT) were used to train the model to detect malapps with 98.5% of TPR(True Positive Rate). In [66], a hybrid model using API calls and permissions was suggested. SVM and RF classifiers were further leveraged to achieve a true positive rate of up to 89%. Authors in [67] presented a model for android malapp detection with intents, permission, and API calls as features and compare the results with four ML classifiers viz. RF, NB, Gradient Boosting(GB), and DT. Accuracy of 96% and TPR of 0.85 were achieved with GB classifier. In [68], authors exploited static and dynamic feature vectors to detect malapps with 89.7% accuracy with a voting classifier-based fusion approach. A hybrid model [69] implemented dynamic analysis on the outcomes of static investigations. API calls and permissions were used as static features, while system calls were used as dynamic features to identify the malapps with an accuracy of 94.6%. Improved Bayesian classifier was used in static analysis and ensemble of

three classifiers viz. RF, GC Forest, and XG boost were used for dynamic analysis. A Hybrid method “MADAM” [50]was proposed to detect the malapps on a rooted device by extracting static and dynamic features. A feature vector was given to K-NN classifier to obtain an accuracy of 96.9%. Dhalaria [8] et al. proposed the hybrid framework by combining the static and dynamic features for the classification of malapps by creating the two feature oriented datasets. This method attains accuracies of 98.53% for malapp detection and 90.10% for its family classification. Kabakus et al. presented a hybrid analysis method for detecting malapps with the maximum detection accuracy of 99.5% when using J48 ML algorithm. Karim et al. [50]proposed a hybrid android smartphone botnet detection platform by exploiting the API calls, permissions as static features and network traffic-based dynamic features for detecting the botnets with high detection accuracy of 98% with RF algo. Ding et al. [70] presented a ResLSTM based hybrid model using static features and traffic based dynamic features to achieve the detection accuracy of 99%. The subsequent section elaborates on the proposed android malapp detection framework. In sum, ML based smartphone security analysis has shown promising results and were in use. But optimal combining of static and dynamic features were limitedly addressed.

Kabakus et al. [71] presented a hybrid analysis method for detecting malapps with the maximum detection accuracy of 99.5% when using J48 ML algorithm. Karim et al. [72] proposed a hybrid android smartphone botnet detection platform by exploiting the API calls, permissions as static features and network traffic-based dynamic features for detecting the botnets with high detection accuracy of 98% with RF algo. Ding et al. [70] presented a ResLSTM based hybrid model using static features and traffic based dynamic features to achieve the detection accuracy of 99%.

In sum, hybrid based smartphone security analysis shown promising results but optimal combining of static and dynamic features were limitedly addressed.

2.5.4 Traffic Analysis

Traffic analysis [73] of smartphone devices also comes under dynamic analysis. Here both the software based and hardware based vulnerabilities can be studied by the analysing the network traffic generated by the smartphones. Coming subsections covers the goals of smart phone traffic analysis and the related work done in this field.

2.5.4.1 Behaviour Analysis: This infers behaviour analysis of smart phone users i.e. user action identification, user fingerprinting and website fingerprinting.

2.5.4.1.1 User Action identification includes identification of a certain action done by the user and inferring information related to certain action performed on his/her mobile device. This analysis helps the researchers in the identification of unknown person in social network and also to build the behavioral profiles which is a source for marketing strategies. Saltaformaggio et al. [74] presented a system called NetScope that can be deployed at network equipment's or Wi-Fi access points for user action identification. It also works in case of IPsec protected traffic because it leverages IP headers/metadata. For evaluating the system, 35 different actions performed on around 22 apps on iOS and Android platform. It achieved average recall and precision of 76.04 per cent and 78.04 per cent respectively.

2.5.4.1.2 User fingerprinting includes detection of the mobile traffic of a particular user. In this analysis, behavioral profile of the smart phone user can also be built by tracing a mobile user using the location of cellular station or Wi-Fi hotspot from the

connected device. Vanrykel et al. [75] proposed a context that involves execution of apps, collection of traffic, inspecting the HTTP data. This also involves searching sensitive identifiers for investigation of user. Such identifiers are exploited for extracting the network traces produced by a particular mobile user from the traffic dataset and hence the authors presented a graph building technique. The analysis over 1260 apps from 42 categories showed that it can relate 57 per cent of mobile users unencrypted network traffic. Additionally, the inadequate efficacy of ad-blocking apps in averting the outflow of sensitive identifiers was also detected.

2.5.4.1.3 Website fingerprinting includes tracking of web pages that a smart phone user visits on web browser. This analysis can be helpful in revealing spiritual faith, concern, lifestyles, along with administrative and voluptuous alignments of user. In [76], Spreitzer et al. developed a fingerprinting technique that uses a Jaccard's index-based ML classifier. The requirement for this technique is only the volume of data processed by which the browser app deals and hence is not affected by encryption.

2.5.4.2 System Identification: This includes identification of apps [77], PII(Personally Identifiable Information) leakage detection [78] and OS identification [79] which are defined as follows:

2.5.4.2.1 Identification of App: To identify the network traffic patterns of specific apps consisting of a behavioral n/w fingerprint that can be figured out in unfamiliar network traces. It is useful for network resource managers in devising the app specific policy forbidding the use of certain sensitive apps (gaming, courting, social networking etc.) in the smartphones by the employees in the company network. In [80],

Mongkolluksamee et al. built an android app identification system focused on combining host-based (communication patterns from graphlets) and statistics based (packet size distribution). Using ML, the evaluation done on real traffic of only 5 mobile apps and gave F-measure of 0.96 by only considering 50 random packets of traffic in duration of 3 min. The main focus of the work was 3G traffic and tcpdump was used to capture the n/w traffic. In [81], Taylor et al. proposed an app identification technique based on ML, called as AppScanner. In this system 110 android apps were fingerprinted from Play Store and gave 96 per cent accurate detection. Further the authors studied that the mobile devices that generate the collected data, network traffic capturing duration and the fingerprinted apps version affects the classification performance. In [82], authors devised a robust approach known as Convolutional neural network based multi-domain learning system that can be employed to identify any VoIP based calling app by fusion of extracted deep features from both temporal domain and spectral domain. The authors in [83] proposed a traffic categorization method to detect smartphones apps producing such traffic (including encrypted traffic) using deep learning methods.

2.5.4.2.2 OS Identification To determine the OS of a smartphone. This analysis serves as initial phase for other complex attacks against smart phone. In this analysis, the adversary attempts to deduce the OS to later exploit known susceptibilities for that particular OS of the target smartphone. In [84], Malik et al. proposed a technique that exploited inter-packet time from a targeted device to deduce its OS. In [85], Coull targeted iMessage, a message service by Apple and tried to conclude the language and size of the texts exchanged among the Apple servers and target users to find if iMessage is being used on OS X or iOS.

2.5.4.2.3 PII Leakage Detection: To analyze the network traffic for detection / preclusion of the outflow of a mobile user's Personal Identifiable Info. PII is used to identify, contact or locate an individual. In [86], authors presented PrivacyGurad for Android platform that was developed by leveraging VPNService class provided by Android SDK for eavesdropping the network traffic of apps. It neither requires any knowledge about VPN nor root permissions from users. The authors employed it for noticing the outflow of sensitive data related to devices and mobile users, such as phone or IMEI (International Mobile Equipment Identity**)** number. The PrivacyGuard is capable of replacing leaked info with bogus data.

2.5.4.3 Malware detection: To detect whether an app behaves maliciously solely on the basis of smartphone traffic. This analysis is capable of performing a security check of the app. In addition to it, the algorithm for malware detection can be rooted into anti-virus apps that can be used further by smartphone users to verify if an installed app is malware or goodware. In [87], Narudin et al. made investigation in order to check if anomaly-based IDS is capable of detecting mal-apps on the basis of traffic analysis. Malware detection is defined as the method using which we identify if an app is malicious or goodware by scrutinizing the network traffic produced by the app. In this section the framework presented by many researchers for Android Malware Identification or detection via traffic analysis is surveyed. An app marketplace , mobile user or security company are mainly interested in malware identification.

The wide spread adoption and success of smart phones has attracted the malware authors. They gain access over confidential data of target device and causes huge danger to users' property. Smart phones have become the prime target of the attackers

due to lots of sensitive info available about the users, its ubiquity and the network capabilities. We surveyed dynamic detection methods based on network-level behaviour which accounts the app behaviour at runtime and find out the leakage of sensitive information and the classification of network behaviour. The studies we cited below involve many facets of malicious smart phone traffic analysis:

TrafficAV [88] is a multi-level n/w traffic analysis approach that used ML and traffic analysis for detection of android malware by effectively detecting malicious traffic. It provides details about detection results, alongside identification of mal-apps. It performs traffic analysis & uncovering on server side, hence not affecting user surfing behaviour and minimal resource consumption of mobile devices. The technology used to gather n/w traffic is traffic mirroring. Further, for data analysis the gathered n/w traffic is directed to a server. It provides user friendly result explanation and offers 2 detection models namely HTTP & TCP flow detection.

NTPDroid [89], a hybrid framework that combined attributes i.e. network traffic and system permissions to create recurrent patterns normal and malware dataset for detection of malicious patterns. The model was trained and tested using FP-Growth algorithm. It has 2 phases i.e. analysis and detection phase. Analysis phase aimed to generate the recurrent patterns present in legitimate and mal-apps. While in detection phase, the 2 sets of recurrent patterns generated are used to identify mal-apps.

In [90], authors presented a dynamic approach analyzing network traffic and capturing app behaviour at run time. Few n/w traffic features were extracted and tested on Decision tree algorithm using WEKA tool. The performance is analysed using accuracy, TPR, FPR (False Positive Rate) , ROC and mean absolute error.

Taking imbalanced learning problem into consideration, authors [91] developed a management and control scheme for collection of android network traffic and established that using n/w traffic to train ML model is a problematic imbalanced learning via analysis of using the collected n/w traffic. In addition, android malwares are detected by applying 4 imbalanced algorithms on imbalanced n/w traffic dataset. It showed that the combination of SVM and SMOTE performed best in all combinations.

Taking into consideration different network flow generation by different apps using different operations and as well as different patterns of both benign and malicious flows, Cheng et al. [92] developed a model using studying of relationship involving behaviour patterns & network flows to detect which app leaks private info of users. It used RF machine learning algorithm for the classification of network flows. Further to improve controllability, authors designed an app called Moledroid [92] to put into practice network flow detection with a ML algorithm. It achieved correctness and exactness higher than 95 per cent.

Watkins et al. [93], demonstrates network-based IDS which detects malware either that generates no network traffic or it is impossible to differentiate the network generated from genuine network traffic. This network-based tool does not engross dependability upon other traffic source from an app and also do not rely on interactions based on host-based dwelling. It does not require its installation on device and is competent to detect a set of malwares which is unable to produce Wi-Fi network traffic. The study [94] focused on identification of mal apps by using URLs visited by apps. This method is responsible for vectorization operations and URL segmentation and does not involve complicated feature engineering. It used neural network having

multiple views and laid emphasis on width & depth of neural network and addressed challenge of feature selection. Multiple views of input are automatically created using this proposed neural network and hence preserved rich semantic info from input and then distributed soft attention weights and its main emphasis is on diverse input features. Wei et al. [95] demonstrated learning and modelling method to analyze android mobile apps network behaviour. To trigger different categories of app behaviours, various app system and environmental factors are simulated. The retrieved series of network event behaviour is classified according to behaviour sequence combination via a Bi-LSTM. Additionally, WGAN viz. Wasserstein Generative Adversarial network was used to solve the trouble of time overhead, limited data samples and hence it increased the diversity of data.

The authors [1] presented a detection method based on text semantics of network flows. It took HTTP flow generated via android apps as documents which is further processed using N-gram method from NLP, to extract text-level features. These features are used to develop a malware detection model. The method uses SVM classifier to find out whether the traffic is malicious or legitimate. It detects unknown samples only when it possesses some characteristics similar to mal samples in training phase. It constructs malware detection model by using N-gram sequence generation, chi-square feature selection algo and SVM algo. Zaman et al. demonstrated a behaviour detection technique for detecting mobile malware that can commune with blacklisted domains and bypass confidential info. For this, App-URL table was created that record all efforts by apps for interacting with remote servers. Each entry in this record takes care for the app id and the URL that app makes contact with. Further, authors used domain blacklist and flagged the apps that make contact with any of the domains as malware. Nancy et al. [77] presented a technique based on network traffic

for detection of malware. The authors compared network traffic of malwares with that of benign apps and found the distinguishing features. A decision tree classifier was built based on the features to detect mal and benign apps. This method achieved accuracy of more than 90% and network traffic of malware was captured using actual smartphones rather than using emulators. In [96], authors analysed the n/w traffic features & built rule-based classifier in order to reveal android malware. Remotely server controlled android malware is detected by it or confidential info of the remote server is disclosed. In the first phase, the method includes analysing network traffic and identifying of distinguishing features among malware and normal traffic. In the second phase, rule-based classifier was built over found distinguishing features and accuracy was calculated by running the classifier on test data and hence it achieved accuracy of 93.75%.

Shabtai et al. [97] presented an anomaly detection method based on behaviour to identify significant variations in device app n/w behaviour. It safeguards cellular infrastructure companies & mobile devices user from mal-apps via detecting mal-attacks and repackaging apps. It attempted to detect malware having self-updating capabilities as this type of malware is not detectable using regular dynamic or static analysis approach. The method uses app network traffic patterns only to perform detection. Arora and Peddoju in [98] focused on minimizing the number of features by proposing an algorithm that prioritize n/w traffic features. Statistical tests are used in this approach to rank the attributes. The end result demonstrated that it reduced training and testing time and as well as gave the high detection accuracy rather using features collectively and hence achieved F-Measure of 0.9636 with 9 features out of total of 22 features. Further, the training time of 300 apps was reduced to 5.8 sec from 11.7 sec and testing time was reduced to 17.3 sec from 25.1 sec for 230 apps. The traffic

was captured on actual smartphone rather than emulator. It is off-device detection and moreover, few samples that use obfuscation technique escape from getting detected by employing encryption remote server interaction.

PODBot [99] is a tool based in cooperation with network traffic analysis and app features. It was assessed over set of botnets of famous types and gives accurate detection of 87 per cent in high threat and 96 per cent in very high threat. By detection via host, bots including which are communicated via Bluetooth or SMS channels are detected but for internet as communication channel, network flow and traffic analysis is effective in detection because of high FPR of static analysis.

In [100], the authors proposed a malware detection arrangement built on TCP traffic that can rapidly and aptly detect malware. Here, n/w traffic produced by several apps has been collected and enormous number of TCP flows resulted after pre-processing. After that early packets size were extracted from TCP flow as features which is then fed to detection model. In this method, the feature extraction time from 53108 network flows is abridged from 39321s to 18041s (drop of 54%). This method also accomplishes a detection rate of 97%.

The authors in [101] suggested an approach grounded on n/w packets fuzzing for Android apps. This system acquire the communication data directed by servers to apps, implements diverse mutation schemes to mutate the different types of novel data, return the mutated response to apps, monitor crash information using log monitoring tools to determine the impending security threats. Four types of problems were exposed by using above approach. The problems comprise unresponsiveness, crashes originated via JSON data exception, URL redirection and HTML content

replacement. The outcomes showed that the suggested technique aptly exposed malign behavior of mobile applications using network information interaction.

Authors in [102], proposed a method for identifying malware based on URLs frequented by the apps. Each URL is divided into various segments using particular characters. The skip-gram algorithm is then used for training. The generated URL vector is then fed into a multi-view neural network based malware detection model. Arora [96] et al. examine the TCP based features based on traffic to shape the classifier for Android malware with more than 90% of detection accuracy. Arora [103] et al. came up with a hybrid model named NTPDroid (Network Traffic and Permissions based Android malapp detection framework) , that uses permissions and traffic features from the apps and exploits a Frequent Pattern Growth algo to generate frequent patterns of permissions and traffic features to achieve detection accuracy of 94%. Wang [103] et al. proposed an efficient malware detection technique by using text semantics of n/w traffic by studying each HTTP flow. These HTTP packets were further processed by NLP (Natural Language Processing) to take out text level features achieving an accuracy of 99.15% but the method achieves 54.81% for unknown apps in the wild. Liu et al. [100]proposed malware detection technique built on TCP n/w traffic, where network traffic generated by apps gets a greater number of TCP flow to extract packet sizes as features. Results achieve 97% of detection accuracy. Ding Li et al. [104] introduced a framework named as DroidClassifier for the identification of HTTP header fields of n/w traffic created by malapps by using a supervised method to train the malware dataset . Moreover, Clustering is also used to increase the classification efficiency. The results achieve 90% of detection accuracy. Li [105] et al. proposed a multilevel detection system named as MulAV, in which it obtains info from n/w traffic, App's source code, geospatial info where n/w traffic is collected by

TCPdump Tool. The info is further fed to ML method to train model which identifies malapps. The result achieves a detection rate of 97.8%. Wang [105] et al. discussed a technique to parse the HTTP packets of n/w traffic where features analyzed are packet avg. length, number of upload and download packets, distribution of packet size etc. Features were further extracted to obtain the pure malicious traffic dataset and this is used to detect malwares.

Su [106] at al.presented Android detection method that uses TCP based behavioral characteristics to detect malapps where capturing of n/w traffic is done using NTM (Network Traffic Monitor] tool and training is done via n/w traffic classifier. Results achieves 99.2% and 94.2% of detection accuracy by using Random forest and J48 classifier. Zulkifli [107] et al. proposed a detection process based on n/w traffic which registers the app behavior and considered 7 TCP based n/w traffic feature from Contagio dumpset & Drebin dataset in which Drebin dataset achieved 98.4% of detection accuracy on J48 decision tree algo.

Malik [108] et al. proposed a pattern based detection method CREDROID which identifies malapps on the basis of the Domain Name Service (DNS) queries, data it transfers to remote server from n/w traffic logs and also protocol used for communication for identifying the credibility of the app. Moreover, Android app can be checked without rooting the android phone. Wang [109] et.al. proposed a malapp detection framework exploiting the URLs(Uniform Resource Locator) visited by them. Here the malapp detection model is based on multi-view neural-network with the detection accuracy of 98.35. Multiple views maintain copious semantic info from inputs for segregating the apps. Wang [110] et. al. suggested a framework for android malapp identification leveraging both the TCP and https features. Here, the app

detection was done at the server side without affecting the user experience. C4.5 ML algo is used to train the model with 8312 benign and 5560 malign apps for identifying unknown apps with accuracy of 97.89%. Sanz [111] et. al. offered a lightweight malapp detection framework using TCP based network features with accuracy of 90% and false positive rate less than 3%. Here total number of 359 malapp and benign apps are used along with two Random forest and AdaBoost ML algorithms.

Alshehri [112] et.al. proposed an innovative method to detect the repackaged apps by investigating the network traffic behaviour of the smartphones. Here authors exploited the request traffic generated by the apps. Total number of 8645 applications were used for experimentation. Here the accuracy of request flows attained is 95.1% and improvement of 18.3% of accuracy when compared with contemporary methods.

Sihag [113] et.al. proposed network packet based investigations of captured traffic of the smartphone. Here, the authors represents the captured network packet interactions as images. These images were given to CNN (Convolution neural network) to achieve detection accuracy of 99.12%.

In sum, security investigations on the smartphone platform via traffic analysis covers identification of software, hardware, OS and app related vulnerabilities. But optimal combinations of various traffic features to create a robust solution for demands further research.

Table 2.3. Comparison of Traffic Analysis Techniques

Author's Name	Methodology	Dataset	Result	Remark
Esmaeili et al. [99]	Based on app features and network traffic analysis KNN, Decision tree, Naive Bayesian as classifier	Drebin, Google Play, Café Bazar	***Precision:*** **high risk**-87 per cent **Very high risk**-96 per cent	Estimation for detection as botnet is done using 3 level of risk i.e. average or high or very high
Zulkifli et al. [90]	Dynamic detection technique Records app behaviour at run time Based on network traffic using Decision Tree	Drebin, Contagiodumpset, Google playstore	Drebin- 98.4% accuracy Contagiodumpset - 97.6%	Drebin achieved higher accuracy in comparison to Contagiodumpset dataset
Arora et al.[114]	Combined network traffic & system permissions Hybrid model	Genome malware dataset	Detction accuracy- 94.25%	Combining permissions and traffic features enhanced the rate of detection
Wei et al. [95]	Modelling and learning method for network behaviour detection Used WGAN and Bi-LSTM	Official Android Market	App classification accuracy- 96.89%	WGAN model helped in improved accuracy of BiLSTM by 9%
S. wang et al. [1]	Based on text semantics of network flows Used N-gram method from NLP SVM classifier	VirusShare [45], Baidu mobile assistant [115], Google play	Accuracy achieved- 99.15%	Requires few samples for good detection results. Able to detect new discovered malware as well.
Kandukuru et al. [116]	Two level hybrid analysis approach based on permission vector and network traffic	Malgenome project, Google playstore	Detection accuracy – 95.56%	Uses less time and limited computational resources

Nepal et al. [117]	Hybrid model based on sensitive resource accessing &network traffic	Google play, Genome malware project, Droidbench	Accurately detected all 3 of app groups of Droidbench dataset i.e Access Internet, Neverclick and sensitive resource	Lower FPR viz. false positive detection rate
Lashkari et al. [118]	Five classifiers namely RF, KNN, DT, RT, Regression Focuses on dynamic behaviour of malware	1900 benign and mal apps of 12 different families	Avg Accuracy (91.41%), Precision (91.24%), FPR (0.085)	Network traffic captured via limited user interaction with installed apps
Pang et al [91]	Analysis of relationship b/wnetwork flows &behaviour patterns.Random Forest ML algo.	Google play [49]	Achieved higher than 95% precision and accuracy	Lacked comparison with other ML methods
Wang et al [88]	Combines network traffic analysis with ML (C4.5 DT) Perform multi-level network traffic analysis	Drebin [53]	***Detection Rate*** TCP Flow – 98.16% HTTP Model – 99.65% ***FPR*** 5.14% and 1.84%	Provides details about detection results
Nancy et al.[77]	Compared traffic of malware with that of normal apps Decision tree as classifier	Android Malware Genome Project	Accuracy – 90.32%	Fails when obfuscation techniques are employed like encrypting the traffic used by malware.
Chen et al.[119]	Involves traffic generation, capturing and behaviour monitoring	Drebin, Android Malware Genome Project	***Detection Rate*** DNS Query: 69.55% HTTP Request: 40.89%	Analysed malware traffic only.
Arora et al. [9]	Based on network traffic features such as ratio of incoming to outgoing bytes, Avg packet size,etc. Uses rule-based classifier	Android Malware Genome Project	Accuracy – 93.75%	The approach is specific to those malwares which in the background connect to any remote server.

Feizollah et al.[120]	Based on network traffic generated by Android apps Uses 2 clustering algorithms i.e. k-means and mini batch k-means	MalGenome, Google play	K-means & Mini Batch K-means : ***Accuracy:*** 0.48, 0.62 ***Homogeneity:*** 0.008, 0.13 ***Completeness:*** 0.16, 0.18 ***V-Measure:*** 0.11, 0.15	Mini batch K-means clustering performed better than the other approach i.e. k-means

2.6 Performance Metrics

To evaluate performance of smartphone security analysis framework, following metrics are normally used:

2.6.1 False Acceptance Rate (FAR) or False Positive Rate (FPR)

FAR [121] or FPR exhibits the probability that malicious application will be treated as benign application. It's also known as 'False Positive' or 'Type I error'. If NM be the total number of malicious applications present and FM be the sum of malicious applications that are incorrectly accepted as benign, then FAR is computed as

$$FPR = \frac{FM}{NM} \tag{2.1}$$

2.6.2 False Rejection Rate (FRR) or False Negative Rate (FNR)

FRR [122] or FNR exhibits the probability that benign application will be regarded as malign application and deprived of installation rights in the smartphone. It's also known as 'False Negative' or 'Type II error'. If FR denotes the total number of benign

applications rejected and NA be total number of actual benign applications, then FNR is calculated as

$$FNR = \frac{FR}{NA} \tag{2.2}$$

2.6.3 Equal Error Rate (EER)

When FAR and FRR are equal then the common value is called EER [123] and is represented as the point at which the plotted curves of FAR and FRR values intersect. EER as shown in Fig 2.2 is also termed as Cross-over error rate between FAR and FRR. This metric expresses the efficacy of the system in rejecting an impostor. If EER is close to zero, then performance of the system is maximum, indicating a clear separation between genuine and imposter.

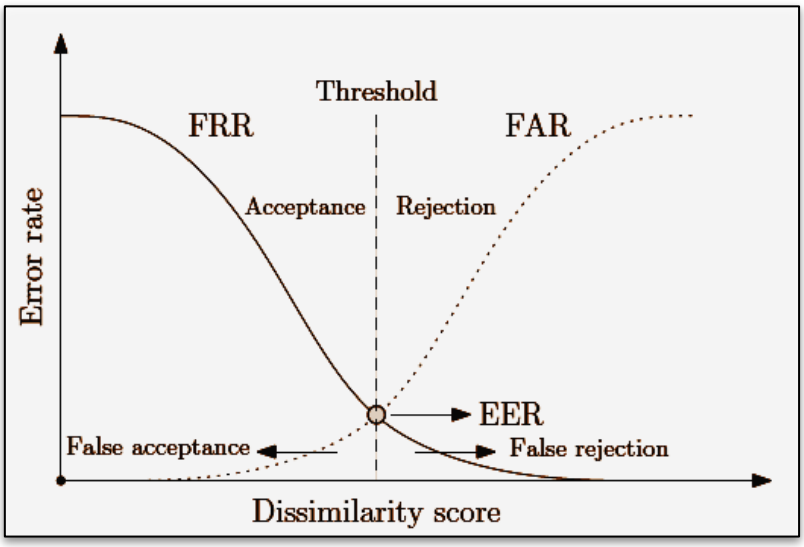

Figure 2.2 Equal error rate (EER)

2.6.4 Receiver Operating Characteristic (ROC)

Receiver Operating Characteristic (ROC) [123] curve is a 2-dimensional plot between False Positive Rate (FPR or FAR) and True Positive Rate (TPR or FRR). In other words, it may be defined as a plot between false match rate against the verification rate. ROC curves as shown in Fig 2.3 is also used to compare the performance of various smartphone security analysis techniques for different threshold values.

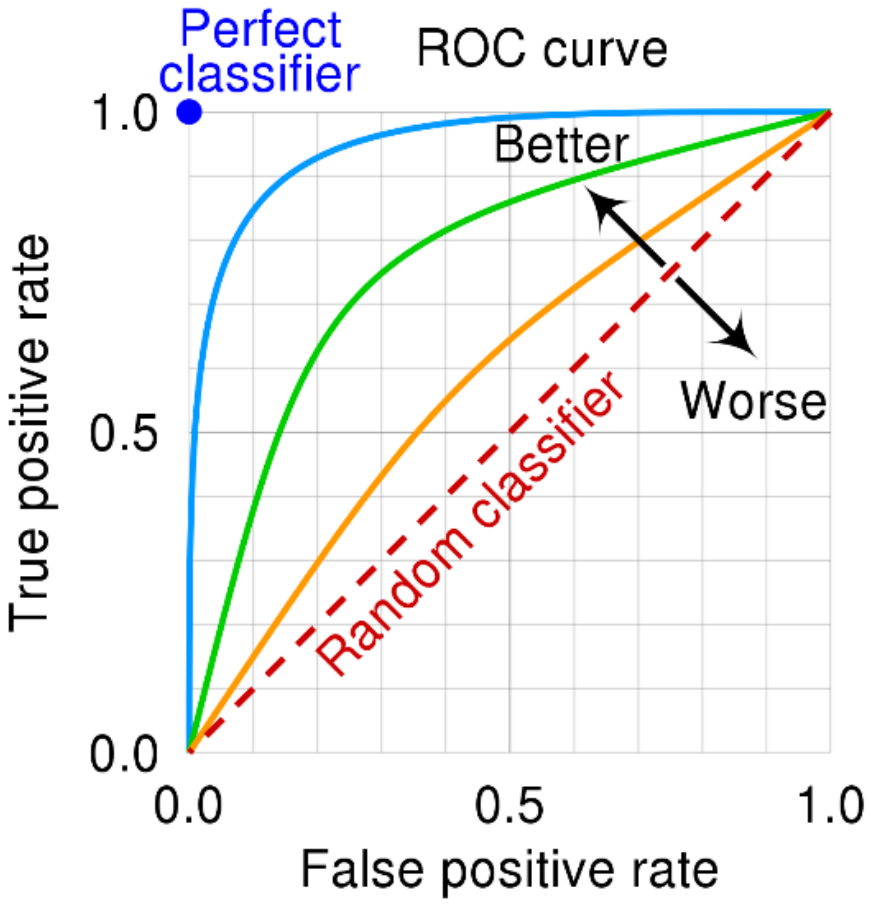

Figure 2.3 ROC curve[2]

[2] https://en.wikipedia.org/wiki/Receiver_operating_characteristic#/media/File:Roc_curve.svg

2.6.5 Decidability Index (D)

Decidability Index (D) [124] determine distance between benign and malicious score distribution. When the malicious application distributions considerably overlap the distributions of the benign applications, it hampers the decision capability of framework.

$$D = \frac{|\mu_B - \mu_M|}{\sqrt{\frac{\sigma_B{}^2 + \sigma^2{}_M}{2}}} \tag{2.3}$$

where, μ_B and μ_M are mean and σ_B and σ_M are variances corresponding of benign and malicious score distributions respectively.

2.6.6 Accuracy

Accuracy [125] measures number of correct predictions to the number of predictions or input samples.

$$DETECTION\ ACCURACY = \frac{TP + TN}{TP + FP + TN + FN} \tag{2.4}$$

2.6.7 Precision

Precision [126] is a metric that is calculated as the ratio of rightly anticipated positives (malicious applications) to the total number of positives (malicious applications) that were anticipated.

$$PRECISION = \frac{TP}{TP + FP} \tag{2.5}$$

2.6.8 Sensitivity or True Positive Rate

Sensitivity [125] is percentage of positives which are correctly recognized by binary classifier.

$$SENSITIVITY = \frac{TP}{TP + FN} \tag{2.6}$$

Also, 1-FRR=TPR

2.6.9 Specificity or True Negative Rate

Specificity [125] is percentage of negatives which are correctly recognized by binary classifier.

$$SPECIFICITY = \frac{TN}{TN + FP} \tag{2.7}$$

2.6.10 F1 Score

F1 Score [127] is weighted mean of sensitivity and precision

$$F1\,SCORE = \frac{2TP}{2TP + FP + FN} \tag{2.8}$$

2.7 Benchmarked Datasets

There are many available datasets for malicious and benign apps

DREBIN [128]: 5560 applications from different malware families.

Malware Genome [43]: 1200 malware samples

AMD [129]:	24,553 malware samples
Google Play [130]:	Over a million benign apps.
Androzoo [131]:	84,420 malware samples
CICMalDroid2020 [132]:	17341 malware samples.

Performance evaluation of a new Smartphone Security Analysis techniques is done on datasets which are publicly available. There are numerous datasets that embed in them the maliciousness of different families of malwares available. Mostly, the malware characteristics of different families are independent of each other. It is very vital to select the right dataset which should be the amalgamation of apps from different sources so as to cover multitude of malwares in the testing and training phase. It is essential for the successful performance evaluation of a malicious app detection system. But sometimes it becomes very problematic to detect the zero day malwares. To detect these malwares, customized dataset incorporating the latest malwares are designed by repackaging techniques. These data sets are then subjected to intensive testing and training on the proposed detection models. Latest malwares in the smartphone ecosystem are collected and embedded in the smartphone apps by repackaging techniques so as to detect the zero day malwares in the smartphone ecosystem.

2.8 Research Gaps

Based upon extensive and intensive review of the literature as discussed in section 2, we have found research gaps in the field of smartphone based mobile platform security. The gaps in the research are mainly concerns with the issues and challenges

in the mobile platform security considering static, dynamic, hybrid, traffic analysis approaches.

Most of the attacks on the malware does not cater to zero-day detection. Future of the malware detection will utilize the online and ensemble learning techniques to detect the malware on the day it comes to the market. New variants of malware are found every day, maintaining the latest patterns efficiently to catch the new trends on usage of permissions is another sustained challenge. Application are utilizing byte code encryption, reflection and native code to thwart static analysis is a great challenge. Automatic reverse engineering of the application to extract the permissions is also a big challenge. Most of the static analysis techniques does not consider the large set of features from the decompiled apps. Source code analysis of the decompiled bytecode is a cumbersome task as it requires more manual analysis.

Automatic generation of multiple static features from decompiled apps and their optimal combination i.e. unification at feature level was not reported in the previous research work. Hence the extraction of complimentary information from large data sets of static features through some efficient technique could be investigated in future study. Most of the work are tested and evaluated on limited set of databases for benign and malign apps. Hence the studies need to be conducted for large sets of data base which are updated with the latest malware and application. In addition, apps available in most of the data sets are generated on an old version of the platform. Also, the malware data base is not uploaded with latest malware. The static and dynamic analysis approaches were investigated separately. There is no little work in literature which consider both static and dynamic analysis to detect malicious behavior of

smartphone-based platform. Categorization of different feature into various threat level was not carried out.

Tuning of critical parameters for optimizing the machine learning models for detection of malwares was not addressed. Optimal combination of different features with automatic parameter tuning and further classification using hybrid approach was not addressed. The malware intentionally prolong the delay of infecting the application to very long time and it is required to monitor the network traffic to an extended period of time to detect the same by network analysis. Most of the malicious activities are restrained due to the constrained environment. Studies has been carried out under the constrained environment which may make the unintended functionality non-operational. Mostly studies are focused on the encrypted traffic generated through mobile apps which may make the analysis cumbersome. Most of the studies has been investigated with the traffic captured from either single apps or few apps. Generalization or adaption of analysis tool is foremost requirement. Considering the various gaps in the research of mobile platform security and also requirement of Defense scenario, I have formulated my research problems and objective of research work.

2.9 Conclusion

Many benchmarked datasets viz. Google Play Store, Drebin, Androzoo, AMD, CICMalDroid2020 etc. are proposed in the literature to evaluate the performance of various smartphone security analysis techniques. In literature, it has been well accepted that combining multiple complementary features extracted from the applications enhances performance and accuracy. Our review work analyze the recent

trends in the field of smartphone security analysis to indicate the future directions to the researchers. Prolific research is going on in this field to detect a reliable malware detection approach. It has been well acknowledged that deep learning based methods provide robust solutions but required large training data and special hardware for their realization. To serve the ever growing demand for training data, large-datasets have also been developed.

Performance of a security analysis of smartphone is measured in terms of performance metrics like False Acceptance Rate (FAR), False Rejection Rate (FRR), Equal Error Rate (EER), Receiver Operating Characteristic (ROC) Curve, Decidability Index, Accuracy, Precision, Sensitivity, Specificity, F1 score etc. Android Malware detection techniques are basically divided into four type(s) viz. static, conventional dynamic, traffic-based dynamic and hybrid analysis. After extensive study, we have found many gaps in the research. The gaps in the research are mainly concerns with the issues and challenges in the mobile platform security considering static, dynamic, hybrid, traffic analysis approaches.

The literature survey resulted in two research papers [133] and [134].

Chapter 3

Design & Development of Static Analysis Technique

The aim of this work is to design and develop a static analysis framework for smartphone security analysis where in a multi stage fusion approach consisting of feature fusion and score level fusion is introduced. For this, a robust unified feature vector is created by fusion of transformed feature matrices corresponding to multi-cue using non-linear graph based cross-diffusion. Unified feature is further subjected to multiple classifiers to obtain their classification scores. Classifier scores are further optimally fused employing Dezert-Smarandache Theory (DSmT) [153].

3.1 Introduction

Static analysis for android malapps detection does not require a host system environment as the apps are not executed. It is also the most economical, proficient and accurate method for investigating the apps. The numerous static features [136] like permissions, app components, filtered intent, API calls etc. are reported in the literature. These features are extracted by disassembling the apps by APK tool19. Permission usage was extensively exploited for development of solution for android

malware detection. Investigations grounded on intents and permissions of applications are susceptible to false positive as benign applications too need sensitive permissions making them misclassified as malapps. Techniques built on only API calls44 frequency are inept to create connections amid the API calls to develop the sophisticated behavioural semantics of apps, leading to poor detection rate of novel malapps. Therefore, choosing multiple complementary features plays a significant part in effective detection of malwares.

Here, an android based Smartphone Security Analysis paradigm has been proposed using non-linear graph fusion and optimal fusion of classifier(s). Multiple complementary features are deduced through extensive investigation of benchmarked datasets. Complementary features are fused through cross-iterative graph diffusion. Thus a unified feature is generated and fed to optimal classifier for classifying the apps into benign or malicious with high detection accuracy. Outcomes of our results demonstrates that proposed method has better performance in classification and detection accuracy. In a nutshell, the following key contributions in this chapter is as follows:

1. We suggest a static feature approach for smartphone security analysis that incorporates multiple feature unification through cross iterative diffusion. To our awareness, it is the first time that this approach is introduced to extract unified android static features.

2. Pragmatic and effective app security analysis framework is proposed wherein three ML algorithms are exploited to evolve a system to detect the malapps on the basis of unified feature representation. Further, outcomes of the ML algorithms were fused by DSmT algorithm to improve the accuracy achieved by individual classifiers. In addition, we presented a complete investigational study based on CICMalDroid2020,

AMD and Drebin malapps database and comparative experimentations with state-of-the-art methods to validate the efficiency and proficiency of our approach.

3.2 Proposed Static Analysis Framework

In this manuscript, multistage fusion model wherein both feature and scores are optimally combined is proposed to achieve highly robust Android malware detection. Overview of the proposed framework is depicted in fig 3.1. For this, we have designed three modules viz. multi-cue feature extraction, feature unification, and optimal classifier fusion to achieve efficient malware detection. For this, semi-automated tool (taking the aid of APK tool) was made to extract the features from the decompiled Android Package Kit. APK tool decompresses the *.apk files into *.dex and AndroidManifest.xml files. Features extracted consists of API calls, Permission, Intents, App Components, Native Code, Op Code, Hardware Feature, Network Address. Each feature is exposed to decision tree learning for generating corresponding similarity matrices. These matrices are subjected to normalization procedure accompanied by filtering out the most weighted similarities in order to generate the sparse matrices. To unify these features, unified-graph is created via graph fusion technique based on cross-iterative diffusion. This method of enhances the robust connections and filters out the weaker ones. Thus, unified set of features are generated for further classification. Unified feature [137] is further subjected to multiple classifier to make final decision about app classification.

Due to multitude of features embedded in the android apps, single ML algorithm presents its inability to classify these apps effectively . Hence, more than one classification algorithms are exploited to detect complex malapp. In the proposed framework, we have chosen three ML algorithms viz. Random Forest(RF), Support

Vector Machine(SVM) and Naïve Bayes(NB) for classification. Also, we proposed optimal combination of these classifier to accurately classify applications into two classes, namely, benign and malicious. Classification algorithms are chosen to compensate the demerits of the individual classifiers. For instance, RF is usually used when there are more number of features than observations. Its performance is excellent in spite of having noise in the predictor variables and it is also not vulnerable to overfitting. Also, RF classifier is preferred for large dataset as it is not susceptible to outliers. On the limited dataset, SVM performance is optimal for two class problem where the data is outlier. For small dataset, NB performs optimally. It is simple and speedy to give classification results. Here, three classifiers complements each other in framework and synergised the performance of resultant classifier. Our approach exploits three classifiers in parallel and the output scores of all the classifiers are fused to synergize the overall performance for detection.

Respective classifier scores viz. S_r for RF, S_s for SVM, S_n for NB are further transformed into belief masses using Shafer Model [138]. Masses for the three class focal elements are optimally combined using PCR-6 Rules [153], where classifier's conflicting mass is redistributed in proportion to the mass which is contributing to the conflict. Finally, in the decision model, belief mass $m_{pcr6}(B)$ is compared with the threshold value m_{thr} and test app is classified into benign depending on whether $m_{pcr6}(B)$ is greater than or equal to the m_{thr} or malign otherwise. Details of the proposed Android malapp detection framework is presented in the next sub-sections.

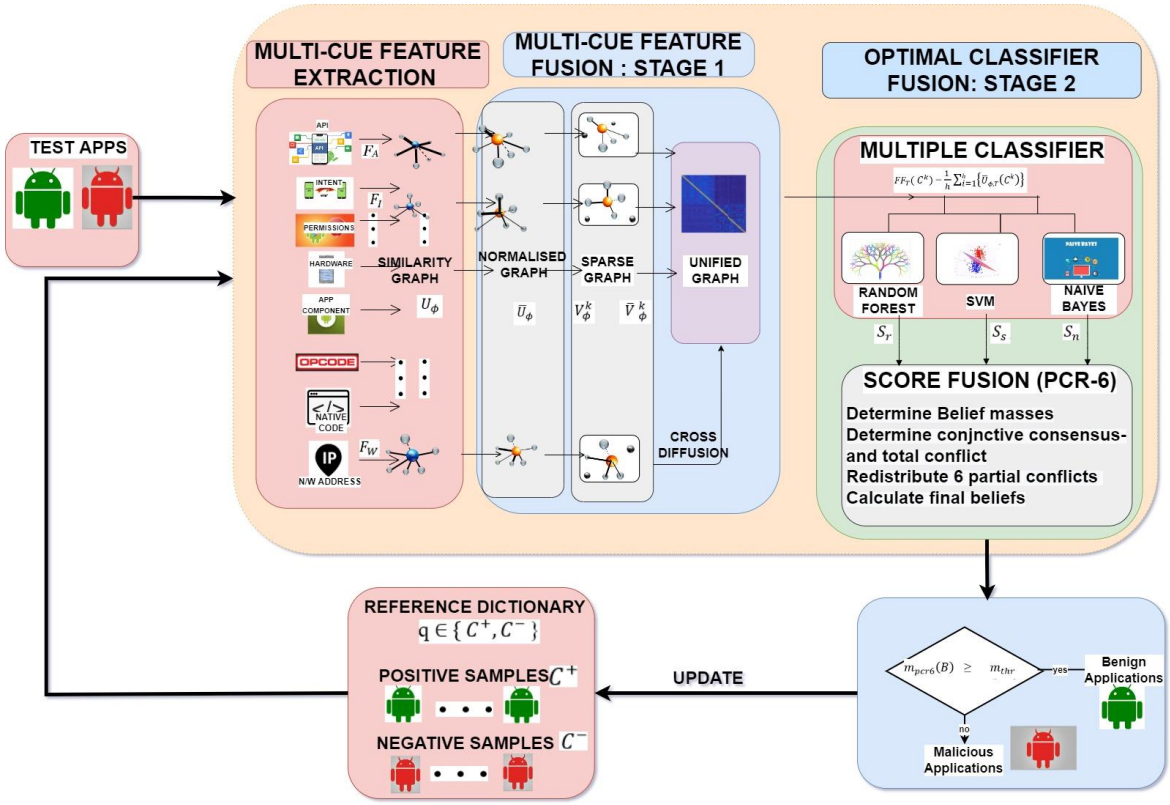

Figure 3.1 Proposed Smartphone Security Analysis Framework. Stage1 extract complementary information from eight multi-cue to obtain unified feature. Unified feature is further classified using optimal fusion of three classifiers at stage 2.

3.2.1 Feature Extraction

Multiple features are extracted for given test app t along with apps from reference dictionary, $q \in \{C^+, C^-\}$, C^+ corresponds to benign apps and C^- corresponds to malicious app. Reference dictionary apps are updated with time so as to incorporate the new apps in the proposed framework to enhance its detection capability. In the proposed approach, we have extracted eight features namely, API calls, Permission, Intents, App Components, Native Code, Op Code, Hardware Feature, Network address using from semi-automated tool.

APIs: Android OS has many APIs (Application Programming Interface) that are used for interacting with Android smartphones. Malwares extensively used APIs to target the Android ecosystem. API's are present in the **.dex* class of an app and can also be found in the Smali Files of the APK. By extensive analysis of the dataset, we have chosen $k1$ number of API's listed in Table **3.1**, whose frequency of occurrence is taken as a feature value. Feature value is determined using Eq. (3.1)

$$F_A^t = \sum_{i=1}^{k1} f(\varphi_{1i}) \tag{3.1}$$

Where, $f(\varphi_{1i})$ is a function that determines the frequency of occurrence of API, φ_{1i} denotes API positioned at i^{th} place in Table **3.1** and F_A^t is the API related feature vector of the test app t . Similarly, API feature F_A^q for reference dictionary apps are extracted for $q \in \{C^+, C^-\}$.

Permissions: Android apps requests for permissions from smartphone user during the app installation. Permissions are essentially required to protect the privacy of the users making the permissions the most vulnerable conduit for launching attacks in the android smartphones. In [24], authors exploited numerous permissions for malware identification. Permissions are stored in Manifest.xml file of the app source code. In our model, we have taken frequency of occurrence of most risky permission's request also as a feature vector. For this, $k2$ number of permissions are chosen considering their frequency of call by various malicious apps. Feature vector related to permission is determined using Eq. (3.2).

$$F_P^t = \sum_{i=1}^{k2} f(\varphi_{2i}) \tag{3.2}$$

.

Where, $f(\varphi_{2i})$ is a function that determines the frequency of occurrence of most frequent permission, φ_{2i} denotes such permission positioned at i^{th} place in Table **3.1** and F_P^t is the permission related feature vector of the test app t .Similarly, permission feature F_P^q for reference dictionary apps are extracted for $q \in \{C^+, C^-\}$.

System Intents: Intent [41] is basically a message used to kick start activity in apps. Starting a service and activity and delivering a broad cast are three basic usage of intents. Malware writers are exploiting intents for launching numerous attacks. We have chosen $k3$ number of intents (listed in Table **3.1**) that are widely used to segregate the malapps from benign apps. Therefore, Intents are taken as feature parameter. Total count of these intents is determined using Eq. (3.3)

$$F_I^t = \sum_{i=1}^{k3} f(\varphi_{3i}) \tag{3.3}$$

Where, $f(\varphi_{3i})$ is a function that determines the frequency of occurrence of Intents φ_{3i}, positioned at i^{th} place in Table **3.1** and F_I^t is the intent related feature vector of $k3$ number of intents of app t .Similarly, Intent feature F_I^q for initially stored apps are extracted for $q \in \{C^+, C^-\}$.

APP Component: App components characterise applications and they are the conduits though which the user or system accesses an app. These components are related by the app's manifest file AndroidManifest.xml that describes the components of an app and dictates the interaction mechanism. There are 4 main app components lying in Android app i.e. Service, Activity, Broadcast Receiver, and Content Provider in the app's manifest file and frequency of these app components are taken as feature parameter and is determined using Eq. (3.4)

$$F_C^t = f(AppComponent) \tag{3.4}$$

F_C^t is the feature value for test app t. Similarly, app component feature F_C^q for reference dictionary apps are extracted for $q \in \{C^+, C^-\}$.

Table 3.1: Details of API, Permission and Intent Feature

Feature	**Position of Feature(*i*)**	**Symbol**
API	AutoSmsReceiver, BootReceiver, PhoneCallReceiver, abortBroadcast, GetCall state, getActiveNetworkInfo(),getDataActivity(),getDeviceId(),getNetworkType(), getSimOperator(), getSimSerialNumber(), getSimState(), getSubscriberId(), classes.dex, entry.loadClass(),getConnectionInfo(), getSupplicantState(), setWiFiEnabled(), execHttpRequest(), Runtime.exec(), Cipher.GetInstance(), sentTextMessage(), getMessageBody(), getSubscriberID(), getLastKnownLocation(), com.android.contacts()	φ_{1i}
Permission	Access_Network_State, set_Prefered_Application, Access_Wi-Fi_State, Access_Fine_Location, Call_Phone, Change_Network_State, Get_Accounts ,Internet, Install_Packages, read_Contacts, Read_Logs, Read_Phone_State, Read_Sms, Receive_Boot_completed, Restart_packages, Receive_Sms,Send_Sms, Vibrate, Write_Secure_Settings, Read_History_Bookmarks, Update_Device_stats, Manage_Documents, Install_Location_Provider	φ_{2i}
Intents	Boot_Completed, Send_To, Dial, Screen_off, Text, Send, User_Present, Screen_On, Call, Package_Data_Cleared, Text, Send, Quickboot_Poweron, Time_Changed, Sms_Received, Airplane_Mode, Battery_Changed Get_Content, Data_Sms_Received	φ_{3i}

Native code: Native code is processor specific code and does not run on the emulator. It is used to hide malicious content in the app as this code is difficult to understand. Therefore, another feature parameter is the total sum of these native codes in an application. Native code feature parameter for app t is calculated using Eq. (3.5)

$$F_N^t = f(NativeCode) \tag{3.5}$$

Similarly, native code feature F_N^q for reference dictionary apps are extracted for $q \in \{C^+, C^-\}$

OP Code: An opcode or Operation Code is a machine language instruction that stipulates the operation to be performed with CPU. Frequency of sequence of opcodes extracted from the apps can be taken as features for malapp identification. Opcodes are exploited by the malware writers because of their similarity to app code and frequency of these op codes are taken as the next feature parameter and is determined using Eq. (3.6)

$$F_O^t = f(OPCode) \tag{3.6}$$

F_O^t is the feature value for test app t. Similarly, op code feature F_O^q for reference dictionary apps are extracted for $q \in \{C^+, C^-\}$

Hardware Feature: Hardware features [135] are used by Android apps to access hardware of the android smartphone and are listed in the AndroidManifest.xml file. Hardware features are characterized by "android. hardware" in the manifest file. In the proposed framework, by extensive analysis of dataset, we have chosen 55 hardware feature for generating feature parameter. The frequency of 55 hardware features in the manifest.xml file of an app is taken as feature parameter and is determined using Eq. (3.7)

$$F_H^t = f(android.hardware) \tag{3.7}$$

F_H^t is the feature value for test app t.Similarly, hardware feature F_H^q for reference dictionary apps are extracted for $q \in \{C^+, C^-\}$

Network Addresses: Malapps designers often wants to interact with malapps so as to direct the user's critical data on the smartphone to designated network addresses of the C&C server embedded in malapps. So, network address can be taken as the feature

parameter. The total number of these network addresses in an application is the network address-based feature parameter for test app t is calculated by Eq. (3.8)

$$F_W^t = f(network\_address) \tag{3.8}$$

F_W^t is the feature value for test app t. Similarly, network address feature F_W^q for reference dictionary apps are extracted for $q \in \{C^+, C^-\}$.

In sum, we have constructed eight feature descriptors as mentioned in Eq. (1) to Eq. (8) for every test application and reference dictionary apps. Similarly, network feature F_W^q for reference dictionary apps are extracted for $q \in \{C^+, C^-\}$. Feature vectors extracted using Eq.(3.1) to Eq.(3.8) are further fused to obtain unified feature. A set of C^+ number of benign and C^- number of malicious apps are stored as a reference dictionary. In feature unification, features for test and reference dictionary apps are extracted and subjected for creation of non-linear graph. In the graph, test app feature act as one node and reference dictionary apps as other nodes. Following this, eight graphs are generated for each test app t.

For each feature descriptor $F_\phi^t \in \{F_A, F_I, F_P, F_C, F_N, F_O, F_H, F_W\}$ of test app t, we construct graphs $U_\phi^t \in \{U_A, U_I, U_P, U_C, U_N, U_O, U_H, U_W\}$ using an edge weight described as the similarity between feature descriptors of two apps t and q where $q \in \{C^+, C^-\}$ .In this, similarity matrices $U_\phi \in \mathbb{R}^{N*N}$ is constructed by decision tree , where $N = q+1$. For feature pair values (F_ϕ^t, F_ϕ^q) corresponding to t and q apps for ϕ^{th} feature, similarity parameter $U_\phi(t,q)$ is calculated by passing them through decision trees using Eq.(3.9)

$$U_\phi(t,q) = \frac{f((L(F_\phi^t)) \cap (L(F_\phi^q)))}{T_t}, \phi \in [A, I, P, C, N, O, H, W] \tag{3.9}$$

L is the set of class labels of trees grown and T_t is the total number of decision trees made. Graph generated using Eq. (3.9) are further fused using proposed cross diffusion to achieve unified feature. Details of features unification follows in turn.

3.2.2 Feature Fusion

Multi-cue feature fusion process consists of multiple feature extraction and their fusion using cross diffusion of extracted features. Multiple features are extracted for achieving the high performance. Cross diffusion of features extract complementary information to obtain highly distinct unified feature leading to creation of clear and distinct boundary between benign and malicious class. In the proposed framework, we have modelled multi-cue feature fusion problem as eight graphs and fused them by iterative cross diffusion process.

Multi-cue features extracted from decompiled source code may not be linearly associated and need non-linear based fusion technique to combine this complementary info. To integrate multi-cues efficiently, non-linear graph based cross-diffusion process was introduced by Wang [139] et al. Further, improved version [140] of this work was explored for classification. In this work, complimentary info from multi-cue data is extracted and non-linear unified graph was generated by cross diffusion process. Classification results of [140] demonstrates that the multi-cue information unification by non-linear graph method is more precise than linear graph methods. Cross diffusion approach proposed by Walia [137] et al. is employed for feature fusion in the proposed framework. This method is better than previous methods [139] [140] and improves the detection accuracy because of the iterative normalization of similarity matrix and updated sparse representation. Unique graph unification approach accomplishes non-linear feature fusion with iterative

normalization. This keeps a robust representation of the apps (malicious or benign) and rejects weak features that make the classifier vulnerable to unreliable results. Similarity matrix generated using Eq. (9) for each feature graph is again normalized using Eq.(3.10) to obtain respective normalized matrices $\overline{U}_{\phi}^{t}, \phi \in \{A, I, P, C, N, O, H, W\}$ .Normalization technique sets the similarity of each app with itself as constant ε, and the similarities with rest of the apps in the test set to (1- ε). The first row of the $\overline{U}_{\phi}^{t}$ comprises the edge weights respective to test app t

$$\overline{U}_{\phi}(t,q) = \begin{cases} (1-\varepsilon)\dfrac{U_{\phi}(t,q)}{(\sum_{q=1}^{N} U_{\phi}(t,q)}, t \neq q \\ \varepsilon, t = q \end{cases} \tag{3.10}$$

$\overline{U}_{\phi}(t,q)$ above is further used to derive a sparse vector depiction of the training app t to keep the most similar features and discard the other using Eq. (3.11)

$$V_{\phi}^{k} = \begin{cases} U_{\phi}(t,q), if\, \overline{U_{\phi}}(t) \in K - NN(t) \\ 0, otherwise \end{cases} \tag{3.11}$$

We further normalized V_{ϕ}^{k} as $\overline{V}_{\phi}^{k}$ vector using Eq. (3.12)

$$\overline{V}_{\phi}^{k} = \frac{V_{\phi}^{k}}{\sum_{i=1}^{N+1} V_{\phi,i}^{k}} \tag{3.12}$$

Normalization further allocates the weights amongst the strong links giving robust sparse depiction. Fused feature descriptor $FF_{T}(t) \in \mathbb{R}^{N+1}$ is obtained by fusing the different feature sparse vectors $\overline{V}_{\phi}$ by cross diffusion approach using Eq. (3.13), where h is the number of features taken.

$$U_{\phi,i+1}(t) = \overline{V}_{\phi,i}^{k} * \left(\frac{1}{h-1} * \sum_{j=1, j\neq\phi}^{h} \left\{ \overline{U}_{\phi}(t) \right\} \right) * \left(\overline{V}_{\phi,i}^{k} \right)^{transpose} \tag{3.13}$$

Where, i is the i^{th} iteration of cross diffusion process and h =8

To enhance the effectiveness of the diffusion, modification of the recursive operation by normalization of each row respective to the test app in the similarity matrix $U_{\phi,i+1}(t)$ $\in \mathbb{R}^{N+1}$ obtained after every iteration as

$$\overline{U}_{\phi,i+1}(t) = \frac{U_{\phi,i+1}(t)}{\sum_{j=1}^{N} \left| \left\{ U_{\phi,i+1}(t) \right\} \right|} \tag{3.14}$$

Following this, normalized sparse vector $\overline{V}_{\phi n,i+1}$ is obtained from sparse vector $V_{\phi n,i+1}$ in the next iteration using Eq.(3.14), Eq.(3.11) and Eq.(3.12). Lastly, the mean of adjacency list of test app for each feature descriptor $\overline{U}_{\phi,T}(t)$ is taken to find the fused feature descriptor $FF_T(t)$ as given in Eq.(3.15)

$$FF_T(t) = \frac{\sum_{j=1}^{8} \left(\overline{U}_{\phi[j],T}(t) \right)}{8} \tag{3.15}$$

Where, T is the final iteration of the normalization process of cross diffusion and $\phi \in \{A, I, P, C, N, O, H, W\}$.This $FF_T(t)$ is taken as a unified feature and given as input to train the classifier(s).Detail of unified feature classification for test app follows in next subsection.

3.2.3 Optimal Classifier Score Fusion

Unified feature for the test app is applied to classification module for final decision. For this we subjected the unified feature to three trained classifiers in parallel to determine their classification scores. Proposed classification model comprises of two phases viz. individual training classifier score estimation and optimal combination of the individual classifier scores. For classification of test app, three classifier scores viz. Random Forest S_r and Support Vector Machine S_s , Naïve Bayes S_n are determined

when unified feature is fed to these individual trained classifier. These classifiers scores can be combined by various score fusion approaches available in the literature [141] [142]. In the proposed method, respective classifier scores attained are further subjected to classifier score fusion paradigm where the obtained scores from the classifiers are converted to respective belief masses and the conflicts amongst individual belief is redistributed and resolved by means of DSmT based PCR-6 rules. Scores of the different classifiers are fused using Shafer's model. For this, the frame of discernment $\left(F_r = \{B, M\}\right)$ is specified by of two focal elements viz. Benign (B) and Malicious (M) corresponding to whether the app is benign or malicious. Each classifier in the model delivers a score about classification. The individual belief mass is obtained by transforming the classifier score $\left(S_r, S_s, S_n\right)$ with the aid of Denoeux Belief System using equations (3.16) and (3.17):

$$m_j(B) = C_j * S_j(B) \tag{3.16}$$

$$m_j(M) = 1 - C_j * S_j(B) \tag{3.17}$$

Where $j \in \{r, s, n\}$ and C_j is the confidence factor of individual classifier. Further, belief masses are optimally fused by means of DsmT-based PCR-6 rules. For this conjunctive consensus is determined using Eq. (3.18) and Eq. (3.19):

$$m_{rsn}(B) = \prod_{j=1}^{3} m_j(B) \tag{3.18}$$

$$m_{rsn}(M) = \prod_{j=1}^{3} m_j(M) \tag{3.19}$$

Where $j \in \{r, s, n\}$ and B, M corresponds to benign and malicious app respectively. Total conflict amongst classifiers is obtained which consists of partial conflicting masses of benign and malicious scores using Eq. (3.20):

$$m_{rsn}(B \cap M = \varnothing) = m_r(B) * m_s(M) * m_n(M) + m_r(M) * m_s(B) * m_n(M) \tag{3.20}$$

$$+m_r(M)*m_s(M)*m_n(B)+m_r(M)*m_s(B)*m_n(B)$$
$$+m_r(B)*m_s(B)*m_n(B)+m_r(B)*m_s(B)*m_n(M)$$

Total conflict comprises of six number of partial conflicts which are further reallocated amongst benign and malicious scores using Equations (3.21-3.26), where $b1$ to $b6$ are redistributed conflict masses for the benign focal element and $m1$ to $m6$ are redistributed conflict masses for the malicious focal element respectively.

$$\frac{b1}{m_r(B)}=\frac{m1}{m_s(M)+m_n(M)}=\frac{m_r(B)*m_s(M)*m_n(M)}{m_r(B)+m_s(M)+m_n(M)} \tag{3.21}$$

$$\frac{b2}{m_s(B)}=\frac{m2}{m_r(M)+m_n(M)}=\frac{m_r(M)*m_s(B)*m_n(M)}{m_r(M)+m_s(B)+m_n(M)} \tag{3.22}$$

$$\frac{b3}{m_n(B)}=\frac{m3}{m_s(M)+m_r(M)}=\frac{m_r(M)*m_s(M)*m_n(B)}{m_r(M)+m_s(M)+m_n(B)} \tag{3.23}$$

$$\frac{b4}{m_s(B)+m_n(B)}=\frac{m4}{m_r(B)}=\frac{m_r(M)*m_s(B)*m_n(B)}{m_r(M)+m_s(M)+m_n(B)} \tag{3.24}$$

$$\frac{b5}{m_r(B)+m_n(B)}=\frac{m5}{m_s(M)}=\frac{m_r(B)*m_s(M)*m_n(B)}{m_r(B)+m_s(M)+m_n(B)} \tag{3.25}$$

$$\frac{b6}{m_r(B)+m_s(B)}=\frac{m6}{m_n(M)}=\frac{m_r(B)*m_s(B)*m_n(M)}{m_r(B)+m_s(M)+m_n(M)} \tag{3.26}$$

The final belief regarding whether the test app is benign or malign is derived by adding the redistribution masses and corresponding conjective consensus using equations Eq. (3.27) and Eq. (3.28)

$$m_{pcr6}(B)=m_{rsn}(B)+b1+b2+b3+b4+b5+b6 \tag{3.27}$$

Algorithm 1: Proposed Smart Phone Security Analysis

Function: Security Analysis $S(t, C^+, C^-)$

For $\left(C^k \in C\right)$ do

S $\leftarrow$ $[C^k, C^+, C^-]$

derive $F_\phi^t \in \{F_A, F_I, F_P, F_C, F_N, F_O, F_H, F_W\}$ from Eq.(3.1-3.8)

for F_ϕ do

Derive $U_\phi^t \in \{U_A, U_I, U_P, U_C, U_N, U_O, U_H, U_W\}$ from Eq. (3.9)

normalize U_ϕ to $\overline{U}_\phi$ using Eq. (3.10)

if $\overline{U}_\phi(t) \in k - NN(t)$ then

$V_\phi^k \leftarrow U_\phi(t, q)$

else

$V_\phi^k = 0$

end

normalize V_ϕ^k to $\overline{V}_\phi^k$ using Eq.(3.12)

repeat

find $U_{\phi,i+1}(t)$, using $\overline{V}_{\phi,i}^k$ and $\overline{U}_\phi(t)$ from Eq.(3.13)

normalize $U_{\phi,i+1}(t)$ to $\overline{U}_{\phi,i+1}(t)$ using Eq.(3.14)

$\overline{U}_{\phi,i}(t) := \overline{U}_{\phi,i+1}(t)$

until convergence

end

find $FF_T(t)$ using $\overline{U}_{\phi,T}(t)$ Eq.(3.15)

find $\left(S_r, S_s, S_n\right)$ using $FF_T(t)$

find $m_j(B)$ and $m_j(M)$, $j \in \{r, s, n\}$ using Eq.(3.16) and Eq.(3.17)

find $m_{pcr6}(B)$ using Eq.(3.18) to (3.26)

find $m_{rsn}(B)$, $m_{rsn}(M)$ and $m_{rsn}(B \cap M = \varnothing)$ from Eq.(3.18-3.20)

find m_j and b_j from Eq.(3.21-3.26) for $j = 1, 2, 3, 4, 5, 6$

find $m_{pcr6}(B)$ using Eq.(3.27)

if $m_{pcr6}(B) \geq m_{thr}$

return (benign)

else

return(malicious)

end

$$m_{pcr6}(M) = m_{rsn}(M) + m1 + m2 + m3 + m4 + m5 + m6 \quad (3.28)$$

The final belief whether test app t is benign or malicious is determined from by $m_{pcr6}(B)$ or $m_{pcr6}(M)$. Thereafter, decision is taken by comparing the final beliefs with a threshold value. If value of $m_{pcr6}(B)$ is greater than or equal to the threshold (m_{thr}) value, then test app t is declared as benign otherwise it is declared as malicious. *Algorithm 1 sum* up the pseudocode for proposed framework for Smartphone Security Analysis. In the next section, performance evaluation of proposed method against other state-of-the-art malware analysis methods follows.

3.3 Experimental Validation

Experimental validation includes both qualitatively and quantitatively evaluation of proposed framework on the chimeric datasets as mentioned in Table **3.2.** Qualitative evaluation is done through statistical investigation of extracted features of datasets and score-distribution of the classifiers. Also, quantitative analysis is done by numerous performance matrices viz. Accuracy, Decidability Index (DI), Equal Error Rate (EER), F1 Score and sensitivity. We also compared our proposed framework with four state-of-art methods employing static features HEMD [19], MLIF [20], DS [141] and FGF [143].The details of the experimental validation follow in turn.

3.3.1 Datasets

For evaluation of proposed framework, Database (DB1 to DB5) comprising of both benign (B) and malign (M) apps datasets is formulated. Benign apps are acquired mainly from Google Play Store [130] and CICMalDroid2020 [132]. After downloading the benign apps, we subject them through online Virus-Total scanner that has about 70 antivirus scanners in its arsenal. Application is tagged as benign if the antivirus scanner recognized it as benign, else it is considered as malign or malicious. Malicious

apps are collected from benchmarked datasets viz. Drebin [128] and AMD [129] and CICMalDroid2020 [132] that covers the diverse families of malware. In total, 4000 apps are collected and rearranged in the form datasets (DB1 to DB5) which is detailed in Table **3.2**. First, four group(DB1-DB4) of 1000 apps each from the benign and malicious apps is created and consolidated group of all the 4000 apps is named as DB5.

Further, evaluation of the framework was performed on MATLAB 2017b on an i7, 2.2 GHz processor having 16 GB RAM to implement proposed framework. In DB1-DB5, we split the dataset of apps into ten equal subsets and select a subset of apps randomly for testing and left over subsets is used as training apps. To overcome over fitting of results, 10 fold cross-validation technique is used and mean values are reported as results. Next section covers the experimental validation where the proposed framework is analysed both in terms of qualitative and quantitative analysis. Particulars of Qualitative analysis follows in the subsequent sub-section.

Table 3.2: Databases for Experimental Validation

App Category / Database	Malign Apps(M)	Benign Apps(B)	Remarks
DB1	500	500	Drebin(M) GooglePlay(B)
DB2	500	500	AMD(M) GooglePlay(B)
DB3	500	500	CICMalDroid2020 (for both M&B)
DB4	500	500	AMD(M) CICMalDroid2020(B)
DB5	2000	2000	Consolidated

3.3.2 Qualitative Analysis

Qualitative performance is evaluated for the proposed framework over the datasets. Qualitative performance is mainly done by comparing the score distribution analysis of different state-of-the-art techniques and frequency analysis of extracted features on the datasets. Qualitative analysis results are deliberated as follows:

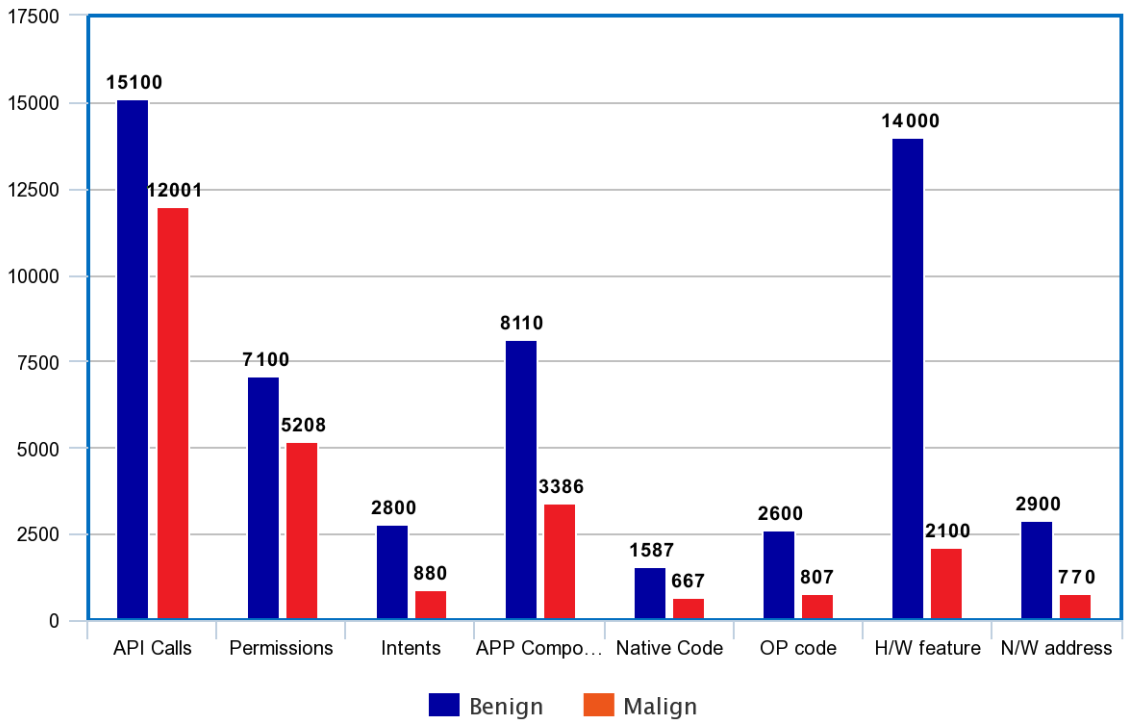

Figure 3.2: Frequency distribution Analysis for extracted Eight features for Database DB5 (x axis : Static Feature ; y axis: Frequency of occurrence of static feature).

3.3.2.1 Frequency Distribution Analysis: Frequency of occurrence of eight complementary features are determined for different datasets. For consolidated dataset DB5, extracted features viz. API calls, Permission, Intents, App Components,

Native Code, Op Code, Hardware Feature and Network Address are plotted as bar charts as shown in Fig. 3.2, for benign and malicious apps, wherein benign apps total feature values are presented in blue colour and malapps total feature values are depicted as red colour bar. From the bar graph, it is apparent that the eight chosen features are discriminative and hence provides a great performance regarding the classification.

3.3.2.2 Score Distribution: To evaluate the proposed optimal classifier performance, score scatter distribution plots are examined for both benign and malicious apps. The outcomes corresponding to DB1 database are shown in Fig 3.3. Scores for benign and malicious apps are determined and plotted against app number resulting in the scatter plot as shown in Fig 3.3.

Fig 3.3(a, b, c, d) displays scatter distribution plot drawn for state-of-the-art methods and the proposed method respectively. From the Fig 3.3, it is clear that most of score are dispersed in the area from 0.4 to 0.6, which is marked as conflicting area. Concentration of apps scores in this range of conflict is maximum for other state-of-the-art methods.

However, using unified feature UF produced by cross iterative diffusion process and proposed optimal classifier, apps scores are broaden as depicted in Fig 3.3(d). Hence, proposed classifier is efficient as it has broadened the classifier(s) score values corresponding to malicious and benign apps.

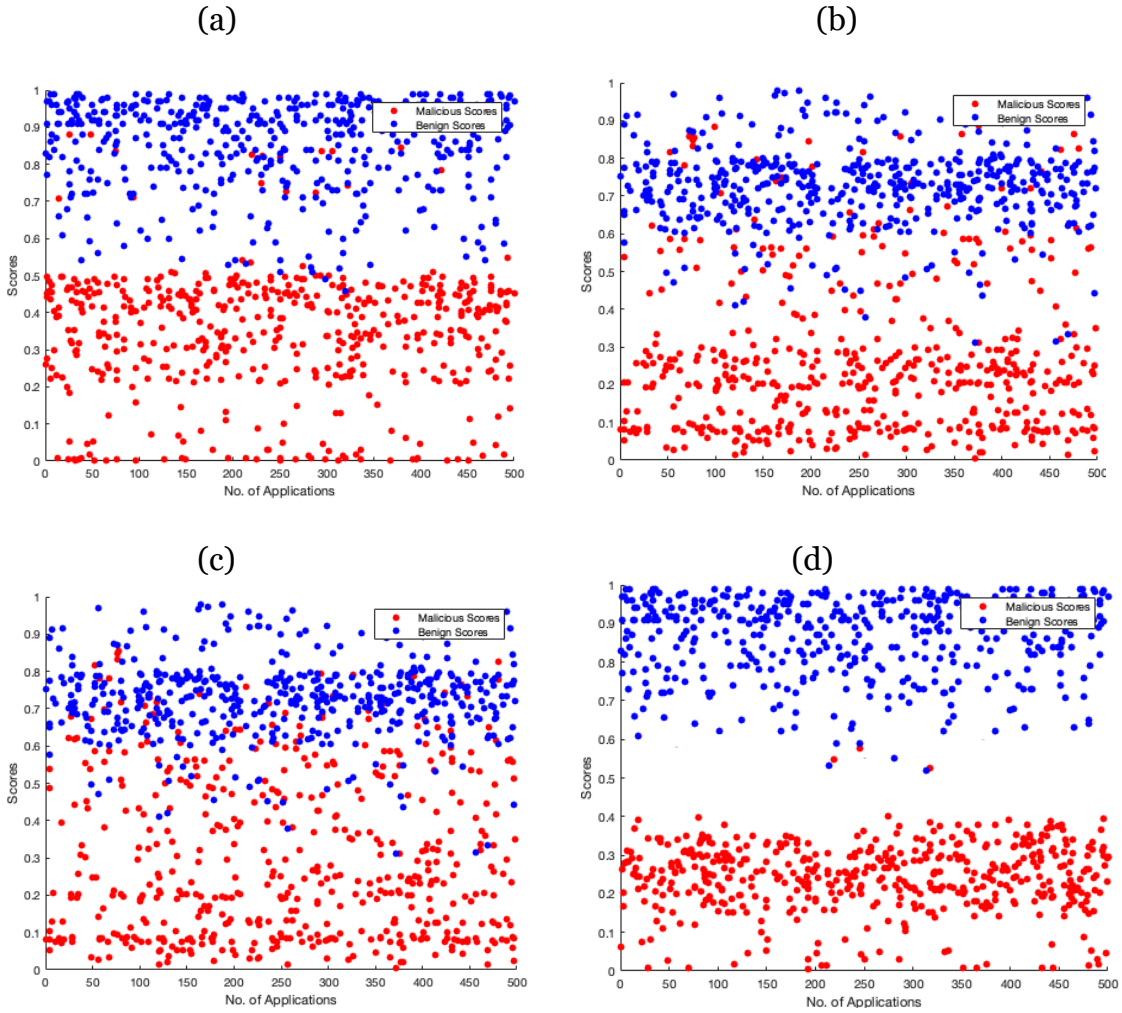

Figure 3.3 Scatter Plots for DB1 dataset:(a) MLIF [20] (b) HEMD [19] (c) DS [141] (d) Proposed Method.

Score for database DB1 are plotted vs frequency of scores value. Overlapping of score values of benign and malicious apps to a large extent render the decision model ineffective. Overlapping of distribution scores occurs for methods [20][19][141] and proposed method as shown in Fig 3.4(a), Fig 3.4(b), Fig 3.4(c) and Fig 4(d) respectively. Minimum overlapping of scores occurs for the proposed method as depicted in the Fig 4(d).

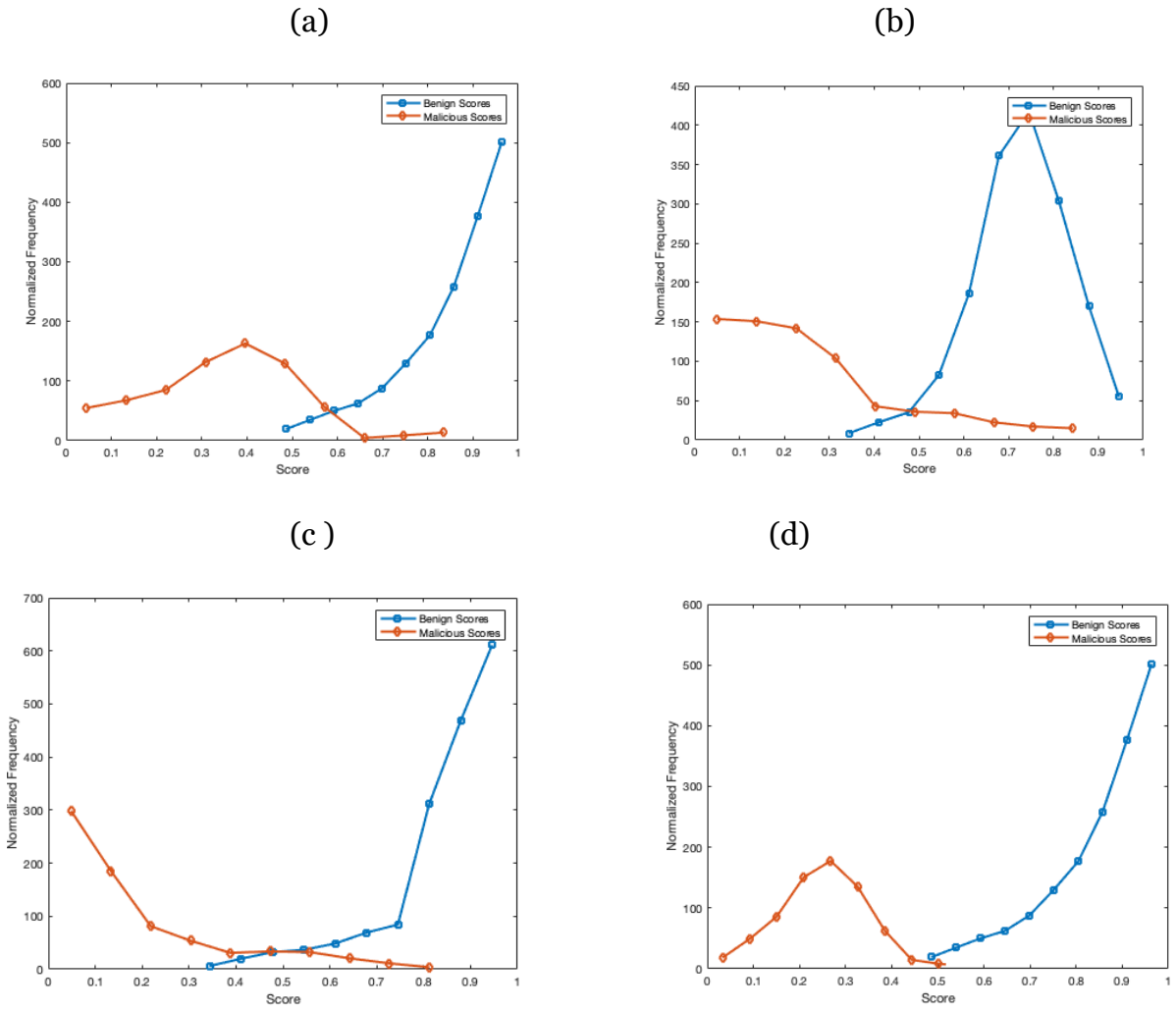

Figure 3.4: Score-Distribution Plots for DB1: :(a) MLIF [20] (b) HEMD [19] (c) DS [141] (d) Proposed Method.

Furthermore, score distribution for the state-of-the-art method and the proposed method are depicted Fig 3.**4**. As shown, score distribution in the proposed multi-stage fusion model in Fig 3.**4 (d),** has minimum overlap. It undoubtedly shows that the distributed scores of the proposed framework can perform better classification. Qualitative analysis further strengthened the Quantitative analysis of proposed framework. Quantitative analysis follows in the next section.

3.3.3 Quantitative Analysis

For the suggested method, quantitative investigation is achieved via ten-fold cross validation on 5 databases (DB1, DB2, DB3, DB4, DB5) of dataset as listed in Table **3.2**. For this, evaluation metrics i.e. Sensitivity, Accuracy, F1 Score, equal error rate and decidability index are calculated and outcomes are compared with the state-of-the-arts methods.

Decidability determine distance between benign and malicious score distribution

Sensitivity is percentage of positives which are correctly recognized by binary classifier. F1 Score is weighted mean of sensitivity and precision. Accuracy measures number of correct prediction to the number of predictions or input samples. Sensitivity, F1 Score and Accuracy are determined using equations as in Section 2.7, Chapter 2.

Decidability index corresponding to database DB1, for various methods are calculated and tabulated in Table **3.3**. Average decidability indexes for are calculated as 2.9544, 3.3551, 2.7934 and 4.10152 for HEMD, MLIF, DS and FGF respectively. Proposed framework attained avg. decidability value of 5.4328 and same is validated by least overlapping of plots in Fig. 3.4(d). This comparatively higher value of decidability of the proposed framework in Table **3.3** is attained largely due to nonlinear feature fusion through cross iterative diffusion and optimal combination of classifiers score.

Table 3.3: Decidability Index for different Methods

Dataset	HEMD	MLIF	DS	FGF	Proposed Method
DB1	2.9699	3.5197	2.5298	4.1384	**5.63**
DB2	2.9167	3.3807	2.8906	4.1264	**4.461**
DB3	2.9373	3.3455	2.9636	4.0071	**5.820**
DB4	2.9671	3.0686	2.7068	4.0814	**5.265**
DB5	2.9812	3.4612	2.8765	4.1543	**5.985**

Receiver Operating Characteristic (ROC) curves have been determined for proposed method, and four state-of-the-art methods. The results are depicted in Fig 3.5. ROC determined the performance of a classifier as its decision threshold is varied. It is evident from the Fig 3.5, for low False Acceptance Rate, proposed method achieves very high False Rejection Rate or in other terms very high true acceptance rate. There is also radical drop in false acceptance rate for state-of-the-art methods. Among ROC curves of the state-of-the-arts methods, method MLIF outperformed methods HEMD, DS and FGF. It is apparent from the ROC curves that proposed framework is extremely precise and proficient.

Proposed method has also been compared with other state-of-the-art methods by calculating the Equal Error Rate (EER) using the ROC curves. Proposed framework achieved very low average EER of 1.0408, whereas other methods attained comparatively higher EER of 7.8562, 3.7800, 8.3026 and 5.9544 for HEMD, MLIF, DS and FGF respectively. Performance enhancement is attributed mainly to the non-

linear graph fusion of eight feature vectors and optimal fusion of classifier scores by DSmT-based proportional conflict redistribution (PCR-6) rules where concurrent scores are enhanced and discordant scores are suppressed.

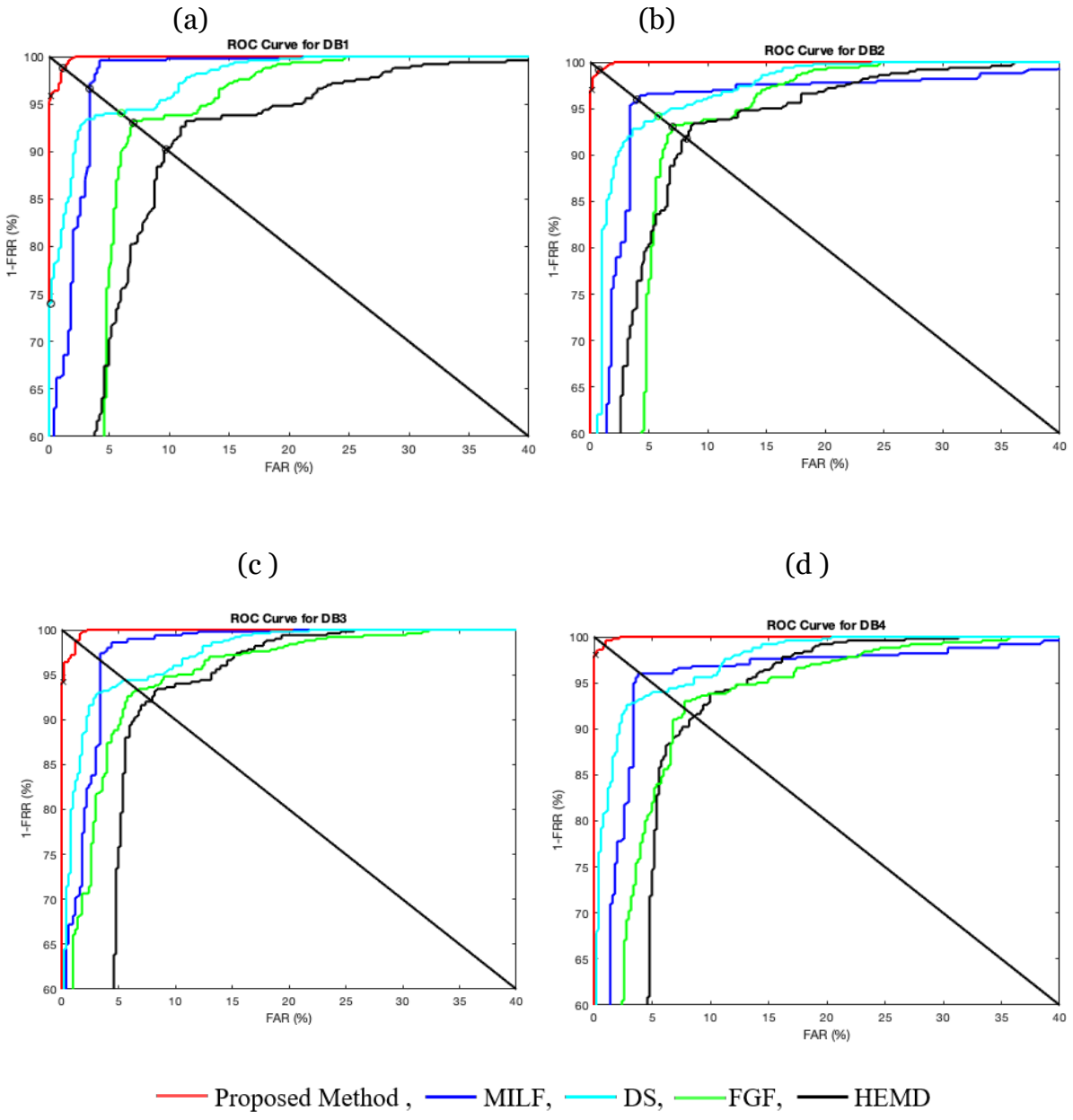

Figure 3.5: Comparison of ROC curves for state-of-the-art method and proposed method (a) DB1, (b) DB2, (c) DB3, (d) DB4

Table 3.4: Evaluation of Average Equal Error Rate for Different Methods.

Dataset	HEMD	MLIF	DS	FGF	Proposed Method
DB1	6.9860	3.4000	9.7804	5.9940	**1.2012**
DB2	8.5828	3.5000	7.0858	5.7942	**0.8008**
DB3	8.1836	3.4000	6.7864	5.9940	**1.1010**
DB4	7.1856	4.4000	8.6260	5.9940	**1.0010**
DB5	8.3434	4.2000	9.2344	5.9962	**1.1000**

In addition, we determine the sensitivity, accuracy and F1 Score for other state-of-art method and proposed method and results are tabulated in Table 3.**5**.

Table 3.5: Comparison of Performance metrics (PM) namely Sensitivity, Accuracy and F1 Score for different comparable methods.

Data set	PM	HEMD	MLIF	DS	FGF	Proposed Method
DB1	Sensitivity	0.9281	0.9660	0.9002	0.9381	0.9880
	Accuracy	0.9271	0.9650	0.9481	0.9380	0.9880
	F1 Score	0.9627	0.9827	0.9474	0.9680	0.9939
DB2	Sensitivity	0.9201	0.9600	0.9102	0.9401	0.9920
	Accuracy	0..9191	0.9590	0.9531	0.9400	0.9919
	F1 Score	0.9584	0.9796	0.9529	0.9691	0.9959
DB3	Sensitivity	0.9122	0.9660	0.9301	0.9381	0.9880
	Accuracy	0.9112	0.9650	0.9630	0.9380	0.9889
	F1 Score	0.9541	0.9827	0.9638	0.9681	0.9939
DB4	Sensitivity	0.9241	0.9640	0.9102	0.9401	0.9900
	Accuracy	0.9231	0.9630	0.9531	0.9400	0.9899
	F1 Score	0.9606	0.9817	0.9529	0.9691	0.9949
DB5	Sensitivity	0.9064	0.9794	0.9583	0.9544	0.9945
	Accuracy	0.9110	0.9650	0.9520	0.9385	0.9898
	F1 Score	0.8220	0.9645	0.9521	0.9360	0.9898

On evaluation over datasets as listed in Table 3.2, avg. detection accuracy of 91.8%, 96.3%, 95.38% and 93.89% for HEMD [19], MLIF [20], DS [141]and FGF[143] respectively has been accomplished. Likewise, average detection accuracy of 98.97% was attained for proposed method. In order to gauge the real time application of proposed method, we have determined time and space complexity of proposed method. On an average, proposed method needs 5.5 sec to evaluate test app. Also, proposed method extracts dimensionality reduced feature as unified feature. Realisation of proposed method is achieved in few KBytes of memory.

Overall performance of the proposed framework versus details of the state-of-the-art methods chosen for comparison, discussed in the next section.

3.2.4 Overall Performance

Proposed smartphone security analysis framework outperforms the other comparable state-of-the-art methods viz. HEMD [19], MLIF [20], DS [141]and FGF [143] both in terms of qualitative and quantitative analysis when evaluated over datasets comprising of benign and malicious apps as tabulated in Table 3.2. Improvement for average accuracy of the proposed method by 7.14%, 2.63%, 3.59% and 5.08% for HEMD, MLIF, DS and FGF respectively has been achieved. An average accuracy of 98.97% was attained for the proposed method. Our framework handles the limitations posed by the state-of-the-art methods by conflict resolution amongst classifiers and redistribution of conflicts to produce improved set of fused scores with better scattering as can be seen in the scattering plots in Fig 3.3. Score distribution plots in Fig 3.4 clearly depicts that the overlapping of malicious and benign scores is reduced to a great extent. Significant improvement in the average decidability index values of proposed model to 5.4328 as compared to 2.954, 3.355, 2.793 and 4.101 for HEMD [19], MLIF [20], DS [141] and FGF[143] respectively further reinforces our claim for

the better performance of the proposed method. This improved value of decidability index of the proposed framework is attained mainly due to nonlinear feature fusion through cross iterative diffusion and optimal combination of classifiers score values. Feature fusion process used in the proposed framework exploits complementary info from the eight individual features.

HEMD [19] uses a set of 4 features viz. permission rate, permission, sensitive API, system events to detect malwares in the apps with a single RF classifier. Here, low accuracy is attributed to lack of feature and score fusion in the model. In Mlif, authors employ parallel machine learning and information fusion approach. Normal vector based feature transformation was employed along with DS theory and probability for malapps detection. Here any conflict arising between the classifier score is not resolved. Authors of DS proposed a multi classifier (SVM, J48, Bayes Net) and fusion method to identify malapps. In FGF, Xu Jianget et. al proposed a static feature (native code, intent filter, reflection, root, permissions) based malapp framework using four ML classifiers (KNN, NB, SVM, J48). This method also achieved low detection accuracy due to lack of feature fusion and optimal classifier fusion.

In a nutshell, the proposed multistage fusion framework for smartphone security analysis outclass other state of the art techniques. It is suitable for classifying test app as malicious or benign with high detection accuracy by feature fusion through cross iterative graph diffusion method and optimal fusion of classifier scores. Quantitative performance enhancement is attributed to extraction of multiple features and their fusion through cross diffusion. Also, our smartphone security analysis framework outperforms numerous limitations of state-of-the-art methods mainly due to extraction of complementary information and optimal fusion of classifiers to create clear and distinct boundary between the benign and malicious classes.

3.6 Conclusion

The significant highlights of this research work are as follows:

- A novel multi stage fusion approach for Smartphone Security Analysis system has been proposed which is founded on the amalgamation of multiple static features and optimal score level fusion.

- The proposed technique is highly efficient to detect the wide variety of malicious apps in the android ecosystem.

- Furthermore, multi-stage fusion stage controls optimum confidence factors for individual classifier. Beliefs of the classifiers are repressed for discordant and enhanced for concurrent classifier(s). PCR-6 rules helps in conflict resolving among classifier beliefs to attain improvement in final score.

- Optimal score fusion applied on cross-diffused features to produce better results than existing state-of-the-art methods.

- An average accuracy of 98.97% was attained for the proposed method. Performance evaluation shows that the proposed method outperforms other state of the art methods. Further, performance metrics viz., EER, Decidability index, ROC curve, F1 score, Accuracy, sensitivity etc. reveals that the proposed method is robust for detection of android malicious applications.

The experimental results along with other findings were published in [145].

Chapter 4

Design & Development of Hybrid Analysis Technique

The aim of this work is to introduce novel hybrid approach for smartphone security analysis. The proposed solution exploits both static and dynamic features for generating a highly distinct unified feature vector using graph based cross-diffusion strategy. Further, a unified feature is subjected to the fuzzy-based classification model to distinguish benign and malicious applications.

4.1 Introduction

Smartphones are deeply rooted in the digital market due to their potential applications in 4G and 5G based wireless networks. Android upheld its status as a leading smartphone OS universally. The profound growth of mobile technology brings significant measures to be incorporated in the mobile security landscape. Also, sum of existing apps in the Google Play repository has been increased to 3.047 million [146]. However, this deluge of mobile apps attracted the malice writers to infuse malwares in these apps for nefarious deeds and the number of new android malwares are also growing as 482579 [147]malware samples per month. Android malicious applications (malapps) proliferate due to the easiness of installing fresh apps from third-party [148]

sources. Amongst different mobile OS, Android is the most widespread platform because of its open architecture. Unluckily, android based smartphones have progressively turned into the key target of the attackers, thereby enforcing urgency for mobile app security. Sailfish OS, Postmarket OS, Ubuntu Touch, Mobian, Lune OS etc are Linux based OS that are also vulnerable to malwares. But due to their limited presence, the attacks are also limited.

Abundant literature is available on static and dynamic analysis to detect malapps and other unintended functionalities in Android apps. These malign (M) apps are normally camouflaged as benign (B) ones causing system impairment, financial damage, information seepage and can form mobile botnets. Numerous investigation mechanism has been suggested to identify malapps. The detection mechanism can be broadly characterized into static and dynamic analysis. Static analysis analyzes code and the manifest.xml file of the app without executing them. However, in dynamic analysis apps are executed and the run-time activities of the apps are analyzed for building solutions. However, static-analysis is thwarted by code-obfuscation and code-polymorphism resulting in variations of malware to escape detections. Dynamic analysis is favourable for analyzing these types of obfuscated apps.

To build a solution to address these issues, static and dynamic features are exploited by the various machine learning (ML) algo to detect the android malwares. The static features mostly used are permissions, app components, intents, API, network address, opcode, hardware component, call flow graph, static taint analysis, dataflow, file property, system command, and native code. The dynamic features frequently used are system calls, API calls, network traffic characteristics, and battery features. In hybrid analysis, both static and dynamic features are exploited for malicious app detection. We propose a hybrid solution that combines both static and dynamic analysis to overcome the limitations of each analysis.

In brief, the key contributions of our paper are concisely described as below:

1. First, we put forward a unique approach for optimal unification of static and dynamic features resulting in Unified feature (UF) for smartphone security analysis by cross diffusion technique.

2. Second, UF is fed to two ML classifiers to detect the android malapps. Results of these classifier's scores were combined by fuzzy based fusion approach for improving the performance.

3. Lastly, we provided a comprehensive study founded on benchmarked database and compare the results with contemporary techniques to validate the efficacy of the suggested framework.

To address the issues in the detection of malapps, we have proposed a novel approach for protection of data. The basics of proposed approach are described in the next section.

4.2 Proposed Hybrid Analysis Framework

In this paper, a hybrid robust unified feature with fuzzy-based optimal score fusion model for android malapp detection is proposed. The outline of the suggested framework is described in Fig.4.1. Our framework is basically comprised of four building blocks namely, feature extraction (static and dynamic vector formation), feature fusion, classifier fusion, and eventually a decision block to attain effective malapp detection. Extracted dynamic features and static features are converted into dynamic and static feature vectors. Each dynamic and static feature vector is used for producing similarity graphs using the cosine similarity. Similarity graphs are further subjected to normalization so as to produce the normalized graphs by filtering out the weighted similarities. Using the reference curves for dynamic and static feature

vectors, we obtain refined graph for dynamic and static feature vectors. The obtained dynamic and static normalized and refined graphs are further cross diffused to produce a diffused graph corresponding to the dynamic and static feature vectors. The diffused graphs of dynamic and static feature vectors are fused to generate a unified feature which is extremely discriminatory. This discriminative unified feature is given to two ML classifiers so as to classify a test app into B or M.

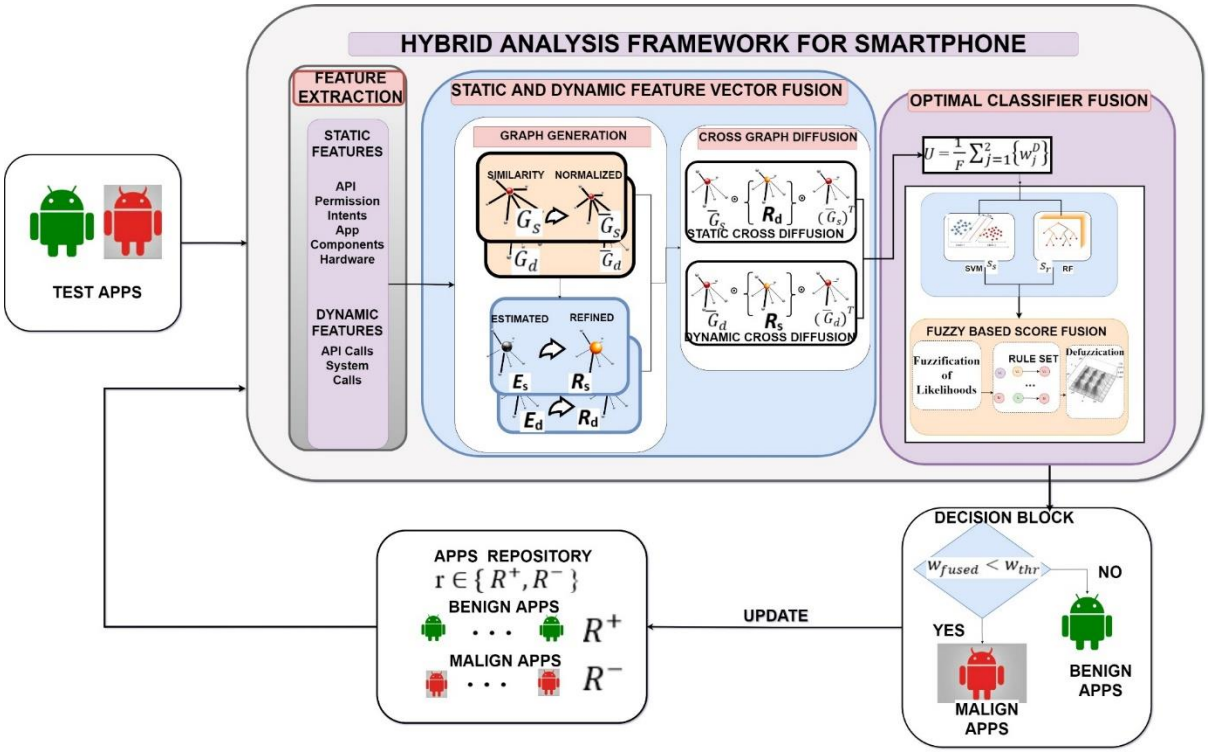

Figure 4.1: Proposed Fusion-based Hybrid technique for Smartphone Platform.

Our methodology exploits two classifiers in parallel whose scores are fused using fuzzy-based fusion technique to enhance the overall performance. Lastly, the final score w_{fused} in the decision model is matched with the threshold, w_{thr} and test app is categorized into B if $w_{fused} \geq w_{thr}$ or M otherwise.

The description of the suggested framework follows in next subsection:

4.2.1 Static and Dynamic Feature Extraction

Description of the eight extracted feature types are as follows:
Feature fusion block comprises static and dynamic feature vector generation after extracting the five static features and collection of two dynamic features. In the proposed model, feature fusion is basically the concatenation of static and dynamic feature vector diffused graphs obtained by cross-diffusion process of normalized and refined graphs.

4.2.1.1 Feature Extraction

Five static and two dynamic features were extracted for a given android test app t together with N android apps from ref. repository, $r \in \left\{R^{+}, R^{-}\right\}$, R^{+} and R^{-} relates to B and M app respectively. The feature extraction process has been illustrated in Fig. 4.2. Apps in the ref. repository are updated so as to include the latest apps to improve the proposed model's detection capability. Here, extraction of static-based features is done using the APK and Baksmali tool. APK tool converts the app into classes.dex and manifest.xml files. Classes.dex files are further subjected to baksmali tool to convert it into smali file. Static-API calls are extracted from smali file. The rest of the static features permissions, hardware features, app components, and intents are extracted from the manifest.xml file. For dynamic features, the system runs the app on a sandbox environment [3] using an Android emulator. The dynamic API calls and the system calls were extracted from the system log files.

(i) Static Features:

API: We have selected $a1$ number of API's whose sum of frequency is the feature value taken. API-linked static-feature vector of test app (t) is computed as follows:

$$F_{AS}^{t} = \sum_{i=1}^{a1} f\left(A_i\right)$$

Where, function $f\left(A_i\right)$ computes the frequency of API, A_i. Likewise, API linked static feature vector F_{AS}^{r} for repository apps are extracted for $r \in \left\{R^{+}, R^{-}\right\}$.

Similarly, pairs $\{F_{Per}^{t} = \sum_{i=1}^{p1} f\left(P_i\right), F_{Per}^{r}\}, \{F_{Int}^{t} = \sum_{i=1}^{i1} f\left(I_i\right), F_{Int}^{r}\}, \{F_{Comp}^{t} = f(App\_Component), F_{Comp}^{r}\}$ and $\{F_{Hard}^{r} = f(Hardware\_Feature), F_{Hard}^{r}\}$ corresponding to Requested Permissions, Intents Filters, APP Component and Hardware Feature were computed.

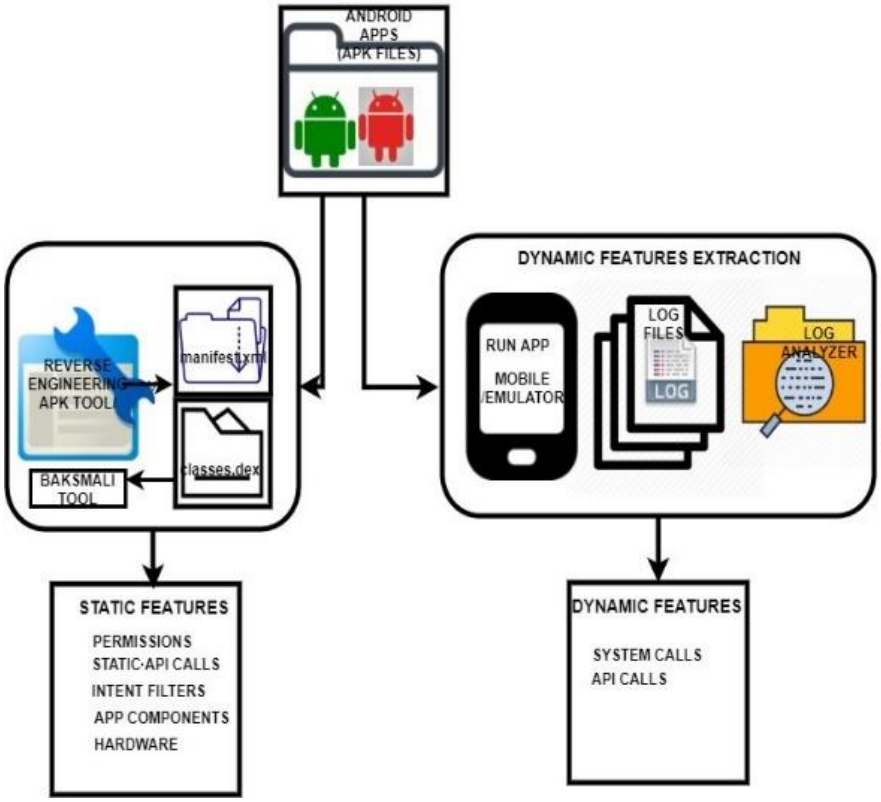

Figure 4.2: Static and Dynamic Feature Extraction Process

(ii) Dynamic Features:

***System calls*:** System call-linked dynamic feature vector of test app t is computed using Eq. (6)

$$F_{Sys}^{t} = f(System\_Calls)$$

Where $f(System\_Calls)$ is a function that calculates total count of system calls resulted by executing the app. Likewise, system calls-linked dynamic feature F_{Sys}^{r} for the repository apps are extracted for $r \in \left\{R^{+}, R^{-}\right\}$.

Similarly, pair $\{F_{Ad}^{t} = f(API\_Calls), F_{Ad}^{r}\}$ for API calls was computed.

From the above seven feature-descriptors, we form two static and dynamic feature vectors as follows in Eq. (4.1) and Eq. (4.2) respectively:

$$F_{Static} = \{F_{AS}, F_{Per}, F_{\text{int}}, F_{comp}, F_{hard}\} \quad (4.1)$$

$$F_{Dynamic} = \{F_{Sys}, F_{Ad}\} \quad (4.2)$$

In short, we have built seven feature-descriptor as stated for every test repository apps. In feature fusion, features vectors corresponding to test and repository apps are used for creating non-linear graph. In the generated graph, feature-vectors corresponding to test apps and repository apps acts as two nodes. Subsequently, graphs are created for each test app t corresponding to static feature vector and dynamic feature vector.

For feature vectors, F_{Static}^{t} and $F_{Dynamic}^{t}$ of test app t corresponding to static and dynamic features, we construct graphs $G_{\phi} = (V_{\phi}, Ed_{\phi}, w_{\phi})$,where $\phi \in \{Static, Dynamic\}$,

w_ϕ are edge weights that act as the similarity between feature-vectors of apps t and r where $r \in \{R^+, R^-\}$, V_ϕ corresponds to the vertices of the generated similarity graphs, Ed_ϕ corresponds to the edges of the similarity graphs that portray the association between the test apps and the repository apps. In the proposed framework, similarity matrices $G_\phi \in \mathbb{R}^{N \times N}$ are constructed by calculating the cosine similarity between the static and dynamic feature vectors of the test app and repository apps, where $N = r + 1$. For feature pair values $\left(F_\phi^t, F_\phi^r\right)$ corresponding to t and r apps, where ϕ corresponds to static and dynamic feature vector. The edge weights are denoted by similarity vector $w_\phi(t,r)$, and is calculated by the cosine similarity between the pair $\left(F_\phi^t, F_\phi^r\right)$ from the following Eq. (4.3)

$$w_\phi(t,r) = \frac{F_\phi^t * F_\phi^r}{\left\|F_\phi^t\right\| \left\|F_\phi^r\right\|} \tag{4.3}$$

Feature unification follows in the coming subsection.

4.2.2 Static and Dynamic Feature Fusion

Constructed static and dynamic feature vectors are fused in a way to extract complementary information embedded in them. This is achieved by the means of the suggested optimal non-linear cross-diffusion of generated refined and normalized graphs to create a distinct borderline between the B and M apps. To unify the multiple features graph-oriented cross diffusion method was presented by [137]. The results validate that the feature fusion via non-linear graph-based technique is better than linear graph-based approaches. Graph-based unification maintains a robust depiction of the apps and discards all the feeble features that contribute to undesirable classification outcomes.

Similarity graph created using Eq. (3) for the static and dynamic feature vectors are again normalized by means of "min-max normalization" to obtain the normalized graphs $\overline{G}_\phi$ whose edge weights are calculated as $\overline{w}_\phi(r)$ using Eq. (4.4)

$$\overline{w}_\phi(t,r) = \frac{w_\phi(t,r) * \min(w_\phi(t,r))}{\max(w_\phi(t,r)) - \min(w_\phi(t,r))} \tag{4.4}$$

Normalized graphs $\overline{G}_\phi$ are employed to obtain the refined graphs R_ϕ, for static features and dynamic features vectors. A refined graph is generated to attain highly distinctive attributes. Normalized attributes are initially deducted out of a generated reference curve to make an estimated graph. There are N normalized weights corresponding to apps used for training. The ideal plot of normalized feature characterized as weight vector $\overline{w}_\phi(t,r)$ can be represented as Eq(4.5):

$$\overline{w}_\phi(t,r) = \begin{cases} 1, \ if \ \ t = r \\ 0, \ otherwise \end{cases} \tag{4.5}$$

Static feature vectors when plotted appear as a curve, where a self-match appeared as a peak and the rest tends to the horizontal line. The more the match score tends to zero, the more dissimilar is the feature value with other apps. Capitalizing this, a curve Re_ϕ representing reference score values is calculated using the training apps, and deviance from Re_ϕ is used to attain adaptability. $\mathrm{Re}_\phi \in \mathbb{R}^{N\times 1}$ is produced by taking mean of the normalized attributes as below by Eq(4.6):

$$\mathrm{Re}_\phi = \frac{\sum_{i=1}^{N} arrange\left[\overline{w}_\phi(t^{\sim}, i)\right]}{N} \tag{4.6}$$

Where $t^{\sim}$ one of training apps of set N and "arrange" is a function used to arrange values in increasing order. This reference curve gives the estimation of training app attributes.

$\overline{G}_{\phi}$ is used to form an estimated graph for test apps, where estimated test app feature components are calculated by deducting $\overline{w}_{\phi}(l)$ from Re_{ϕ} by Eq.(4.7) as below:

$$w_{\phi}^{e}(l)=\begin{cases}\overline{w}_{\phi}(l), & l<m\\ \left|\overline{w}_{\phi}(l)-\mathrm{Re}_{\phi}(l)\right|, & m\le l\le N\end{cases} \tag{4.7}$$

Where, $w_{\phi}^{e}\in\mathbb{R}^{N\times 1}$ denotes estimated test app attribute and the variable m segregate the dimension of a feature vector. This method helps in generating the highly discriminative test app features leading to the detection of the malapps with high efficiency. Estimated features $w_{\phi}^{e}(l)$ are plotted to determine the test app's estimated feature weights. Significant area under curve (SA) of the estimated feature plot is determined and its weight e_{ϕ} is calculated using Eq.(4.8).

$$e_{\phi}=N\times\left[\frac{\frac{1}{\left(SA_{\phi}\right)_{t}}}{\sum_{i=1}^{N}\frac{1}{\left(SA_{\phi}\right)_{i}}}\right] \tag{4.8}$$

Where, $\left(SA_{\phi}\right)_{i}$ and $\left(SA_{\phi}\right)_{t}$ are the SA of the i^{th} app used for training and of the test app t in the estimated graph respectively. This area signifies feature efficacy.

To generate a Refined Graph R_{ϕ}^{t} , its edge weights $w_{\phi}^{r}(l)$ are calculated by reorganizing $w_{\phi}^{e}(l)$ by means of Eq.(4.9). R_{ϕ}^{t} resulted in the robust connections among vertices of graph and all the feeble connections are significantly reduced.

$$w_{\phi}^{r}(l) = \left(w_{\phi}^{e}(l)\right)^{\alpha e_{\phi}} \tag{4.9}$$

Where α is pre-estimated constant used to achieve adaptiveness. Normalized and refined graphs for static features and dynamic features vectors are further exploited in the cross diffusion process.

Static features and dynamic features vectors possess distinctive and complementary information for the segregation of apps into B or M. Therefore, the cross-diffusion of static feature vector normalized graph and the dynamic feature vector refined graph and vice-versa results in boosting up of the robust connections along with filtering out the weaker connections leading to better accuracy.

In the proposed framework, $\overline{G}_{\phi}$ and R_{ϕ} are fused via non-linear cross diffusion scheme resulting in the fused graphs D_{ϕ} with edge-weights calculated using Eq.(4.10)

$$w_{\phi}^{diffused} = \overline{w}_{\phi} * \left(\frac{2}{F} * \sum_{j=1, j\neq\phi}^{F} \left(w_{j}^{r}\right)\right) * \left(\overline{w}_{\phi}\right)^{T} \tag{4.10}$$

Where F is the total number of feature vectors and T above represents transposition. In our framework, $F = 2$ as we have taken only two feature vectors i.e. static and dynamic feature vector.

The diffused edge weights $w_{\phi}^{diffused}$ are further used to form a unified feature vector U by means of Eq.(4.11).

$$U(t) = \frac{1}{\beta F} * \left(\sum_{j=1}^{2} \left(w_{j}^{diffused}\right)\right) \tag{4.11}$$

Where $j = 1$ corresponds to static feature vector, $j = 2$ corresponds to dynamic feature vector and β is the pre-estimated constant used to achieve adaptiveness. Details of the classifier fusion follows next.

4.2.3 Fuzzy Based Score Level Fusion

Vector U generated is given to the two classifiers in parallel. Classification scores obtained are again fused. To achieve this, the U is inputted to two ML classifiers viz. SVM and RF. Their respective classification scores (SVM) and (RF) are determined. In the proposed framework, the obtained classifier(s) scores are optimally fused using the fuzzy-based score fusion method. In the proposed method, a fuzzy-based score-fusion method has been suggested to improve the segregation of apps. Fuzzy [149] fusion is basically combining scores of two ML algorithms in a natural way to determine valuable info and to boost the performances of the individual algorithm. In the suggested method, the fuzzy logic conditions are formulated by a group of twenty-five fuzzy rules as stated in Tab.1 under Section 3.2. The classifiers scores are combined in a way so as to boost the concurrent classifier scores, suppress the discordant classifier scores. The proposed fusion model attains a precise decision boundary between B and M apps. Here, we have defined the fuzzy set as signifying very large, large, medium, small, very small values of the classifier's score. Membership value for the classification score is calculated as elements of a fuzzy set by means of Eq. (4.12-4.16)

$$\Psi_{VL}(x) = \frac{1}{1+e^{-m_{VL}(x-f_{VL})}} \tag{4.12}$$

$$\Psi_{L}(x) = e^{\frac{-(x-n_L)^2}{f_L^2}} \tag{4.13}$$

$$\Psi_{M}(x) = e^{\frac{-(x-n_M)^2}{f_M^2}} \tag{4.14}$$

$$\Psi_{S}(x) = e^{\frac{-(x-n_S)^2}{f_S^2}} \tag{4.15}$$

$$\Psi_{VS}(x) = \frac{1}{1+e^{-m_{VS}(x-f_{VS})}} \tag{4.16}$$

Where $m_{VL} = 36$, $f_{VL} = 0.84$, $f_L = 0.13$, $n_L = 0.7$, $f_M = 0.13$, $n_M = 0.5$, $f_S = 0.12$, $n_S = 0.3$, $f_{VS} = 0.14$, $m_{VS} = -36$ are linguistic variable values attained from the training phase and $x \in \{S_r, S_s\}$.These values are chosen so that concordant classifiers scores are boosted and discordant classifier scores are suppressed concurrently. The functional mapping $T_{u,v}$ between the RF and SVM classifiers scores are tabulated in Table 4.1, where u and v are fuzzy set values allocated to each score value. This mapping guarantees an accurate decision boundary-line for segregating the malapps.

Table 4.1: Fuzzy Mapping Rules $T_{u,v}$

u \ v	VL	L	M	S	VS
VL	VL	VL	L	L	L
L	VL	VL	M	M	M
M	L	M	M	S	VS
S	L	M	S	VS	VS
VS	L	M	VS	VS	VS

Fuzzy fused output is further converted to the optimal crisp value using the center of gravity (COG) technique for defuzzification [150]. Crisp variable value using COG for a pair of elements u and v in the fuzzy set is calculated using Eq. (4.17)

$$COG_{T_{u,v}} = \frac{\sum_u \sum_v \Psi_{T_{u,v}}(x)x}{\sum_u \sum_v \Psi_{T_{u,v}}(x)} \tag{4.17}$$

where value of $T_{u,v}$ is taken from Table 4.1 and $x \in \{S_r, S_s\}$. The weighted mean of the COG values over the pair of elements u, v is used for the estimation of the final fused weight of the classifiers scores [Eq.(4.18)]

$$w_{fused} = \frac{\sum_u \sum_v \Omega_{u,v} * COG_{T_{u,v}}}{\sum_u \sum_v \Omega_{u,v}(S_r, S_s)} \tag{4.18}$$

where $\Omega_{u,v}$ is fuzzy control rule calculated using Eq.(4.19)

$$\Omega_{u,v} = \min(\Psi_p(S_r), \Psi_q(S_s)) \tag{4.19}$$

The w_{fused} is compared with the w_{thr} to determine whether a given app is M or B depending on whether $w_{fused} < w_{thr}$ or vice-versa ($w_{fused} > w_{thr}$). Experimental results and discussions follows in the subsequent subsection.

4.3. Experimental Validation

Experimental results comprise of the assessment of the suggested framework on the benchmarked datasets containing (B) and (M) apps as mentioned in Table 4.2. Comparisons of the results with other state-of-art methods employing static and dynamic features were also reported.

4.3.1. Datasets

B apps are taken from CICMalDroid2020 [132] Dataset and Google Play Store [130] and M apps are collected from CICMalDroid2020 [132], AMD [129], Androzoo [131], and Drebin [128]covering the multitude of malwares from different families. 2000 B and 2000 M apps are selected from these datasets and rearranged in Table 4.2 as Group1, Group2, Group3, Group4, & Group5. The balanced and unbalanced datasets can be used for experimentation purpose. Here, we have used balanced datasets for experimentation.

Table 4.2: Experimentation Dataset

App kind / Dataset	Malign Apps(M)	Benign Apps(B)	Comments
Group1	500	500	Androzoo(M) [131] GooglePlay(B) [130]
Group2	500	500	AMD(M) [129] CICMalDroid2020(B) [132]
aGroup3	500	500	CICMalDroid2020(M) [132] GooglePlay(B) [130]
Group4	500	500	Drebin(M) [128] GooglePlay(B) [130]
Group5	2000	2000	Consolidated

Experimental validation of the framework was accomplished by means of MATLAB 2018a installed on i7, 2.7 GHz CPU with 16 GB RAM. Ten-fold cross-validation method was employed by randomly subdividing the dataset into ten equal parts and using one for testing and the rest for training. The final result is the average of the results obtained from five datasets as in Table 4.2.

4.3.2 Qualitative Assessment

Cumulative Frequency Analysis: Qualitative assessment for the suggested framework is performed by plotting the cumulative frequencies (CF) [as shown in Fig 4.2] of the various static and dynamic features selected for M and B apps. The CF of the features in the M apps is directly proportional to the threat level of that particular feature for platform security. Also, score distributions of the two best state-of-the-art techniques and the suggested framework is shown in Fig.4.4. It is apparent from Fig.4.4 that the suggested framework performs better than the other two methods

because of the minimum overlap of the scores. Quantitative assessment of the proposed framework follows in the sub-section.

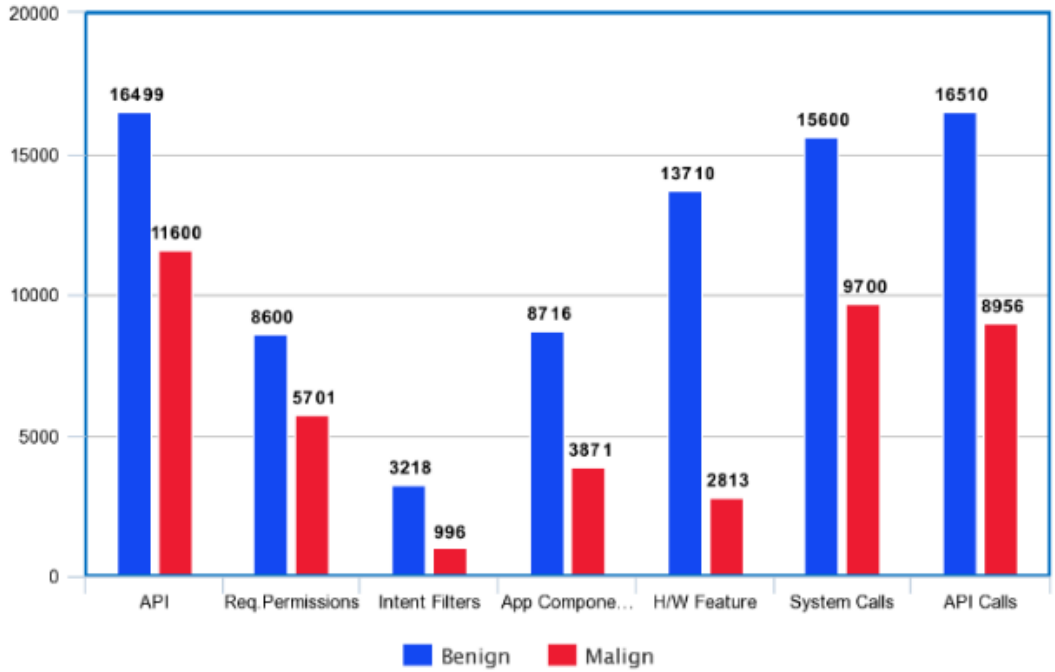

Figure 4.3. Cumulative Frequencies for five static and two dynamic features for Group 5 Dataset. Y-axis represents the cumulative frequencies of features.

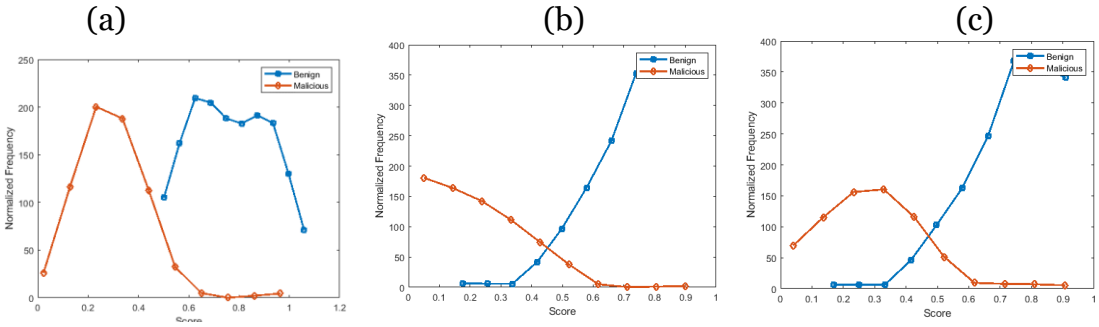

Figure 4.4 Score Distribution for Group2 dataset of (a) Proposed Method (b) Arshad et al. [60] (c) Hussain et al. [67]

4.3.3 Quantitative Assessment

Quantitative assessment of the suggested method was realized using the standard evaluation benchmark viz. sensitivity, specificity, F1 Score, detection accuracy via ten-fold cross validation over datasets as mentioned in Table 4.1. The suggested method

was also compared with respect to running time against different state-of-the-art methods. Evaluation matrices results are also compared with the two state-of-the-arts techniques viz. [67][60] and two self-proposed techniques. Specificity, Sensitivity, F1 Score, and Accuracy are calculated using equation as in Section 2.7 in Chapter 2.

ROC curve is also drawn as depicted in Fig 4.5 to assess the binary classifier. ROC is an overall index portraying sensitivity and specificity.

To test the robustness and to evade overfitting issues, 10-fold cross-validation is employed to estimate the performance of the suggested model. The investigational outcomes are displayed in Table 4.3.

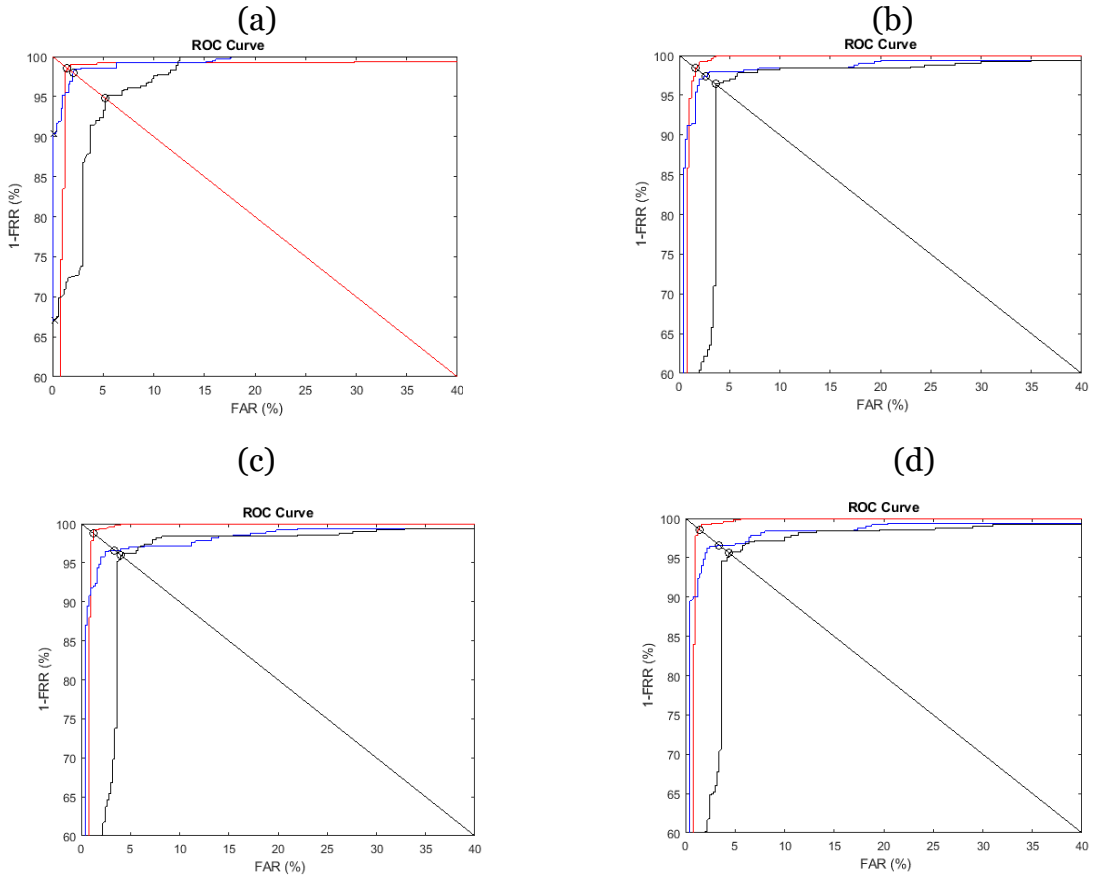

Figure 4.5: ROC curves comparison for the proposed method (red) and two best state-of-the-art methods (Arshad et al. (blue) and Hussain et al. (black)) for (a) Group1, (b) Group 2, (c) Group 3, (d) Group 4.

4.3.4 Overall Performance

It has been observed that for the suggested framework the mean value of the result of the Accuracy, Specificity, Sensitivity, and F1 measure for the proposed framework are 98.62%, 98.634%, 99.30%, and 0.9916 respectively. The maximum value of the accuracy, specificity, sensitivity, and F1 score is 98.80%, 98.80%, 98.81%, and 0.9940 respectively.

The suggested technique outperforms the other hybrid-based state-of-the-art techniques when assessed on datasets as tabularized in Table 4.2. Enhancement for a mean value of detection accuracy of the suggested technique by 1.402%, 2.914%, over [60], and [67] respectively have been realized. Self-proposed techniques are also included to show the proper justification for the choice of ML algorithms in the optimal classifier. The enhancement of average detection accuracy of 2.674% and 3.274% have been achieved by the proposed method over two self-proposed methods RF+UF and SVM+UF.

The run time of different state-of-the-art approaches is also compared with the proposed approach. To calculate the running time of different methods, we first built and learned their corresponding detection model. These detection models were then fed with the 200 random apps for analysis. Our proposed method attains an amazing average analysis performance of 5.6 seconds per app. Similarly, the average analysis performance of [60], and [67] comes out to be 6.1 seconds and 6.7 seconds respectively. Hence, our proposed method outclassed other methods in respect of detection time, detection accuracy, and efficacy in real-life apps scenarios.

Table 4.3: Comparative Analysis of Performance metrics i.e. Accuracy, Specificity, Sensitivity, F1 Score for Hybrid models and Proposed method.

Dataset	Performance Metrics	Arshad et al.	Hussain et al.	RF+UF	SVM+ UF	Proposed Method
GROUP 1	Accuracy	0.9781	0.9480	0.9580	0.9540	0.9860
	Specificity	0.9786	0.9481	0.9560	0.9560	0.9859
	Sensitivity	0.9775	0.9480	0.9600	0.9520	0.9861
	F1 Score	0.9886	0.9733	0.9581	0.9539	0.9930
GROUP 2	Accuracy	0.9760	0.9640	0.9590	0.9530	0.9840
	Specificity	0.9759	0.9641	0.9600	0.9520	0.9839
	Sensitivity	0.9757	0.9639	0.9580	0.9540	0.9842
	F1 Score	0.9869	0.9817	0.9590	0.9530	0.9919
GROUP 3	Accuracy	0.9670	0.9600	0.9600	0.9520	0.9880
	Specificity	0.9680	0.9600	0.9660	0.9540	0.9878
	Sensitivity	0.9660	0.9600	0.9540	0.9500	0.9881
	F1 Score	0.9827	0.9776	0.9598	0.9519	0.9940
GROUP 4	Accuracy	0.9660	0.9560	0.9610	0.9550	0.9860
	Specificity	0.9659	0.9558	0.9640	0.9560	0.9861
	Sensitivity	0.9661	0.9561	0.9580	0.9540	0.9858
	F1 Score	0.9827	0.9775	0.9609	0.9550	0.9930
GROUP 5	Accuracy	0.9738	0.9573	0.9593	0.9533	0.9870
	Specificity	0.9650	0.9540	0.9600	0.9540	0.9880
	Sensitivity	0.9825	0.9605	0.9585	0.9525	0.9860
	F1 Score	0.9740	0.9574	0.9592	0.9532	0.9870

4.4 Conclusion

The significant highlights of this research work are as follows:

- A novel hybrid framework has been proposed for smartphone security analysis.
- Here, the optimal combination of static and dynamic features by cross-diffusion followed by fuzzy-based score level fusion was proposed.
- In the suggested framework, we have used five static and two dynamic features to form static and dynamic feature vectors. These feature vectors are further fused after the formation of normalized and refined graphs through the non-linear graph diffusion method.
- The fused feature vector is then given to an optimal classifier comprising of RF and SVM classifiers. A remarkable benefit of our method is that it can extract the static and dynamic features in each app almost in real-time. In sum, the unification of static and dynamic feature achieve highly distinct features.
- Adoption of fuzzy-based fusion of classifier scores not only create clear boundary but also achieve optimal performance.
- Our technique has accomplished mean value of accuracy, specificity, sensitivity, and F1 score as 98.62%, 98.634%, 98.604 %, and 99.16% respectively when estimated on datasets as enumerated in Table 4.2. Experimental results reveal that our technique outstrips other state-of-the-art methods.

The experimental results along with other findings were published in [151].

Chapter 5

Design & Development of Traffic Analysis Technique

The aim of this work is to develop a novel traffic-based framework that exploits the TCP based network traffic features for the detection of unintended functionality has been proposed. Here, a unified feature (UF) is created by graph-based cross-diffusion of generated order and sparse matrices corresponding to the network traffic features. Generated unified feature is then given to three classifiers to get corresponding classifier scores. The classifiers score are further fused by score fusion process to detect the unintended functionality.

5.1 Introduction

Smartphones are replacing the conventional mobiles as well as computational devices due to its portability and ease of handling almost everything ranging from storing the private data to making the banking transactions. Due to its widespread Smartphones are replacing the conventional mobiles as well as computational devices due to its portability and ease of handling almost everything ranging from storing private data to making banking transactions. Smartphones having Android OS are extensively familiar and have wide usage due to its open architecture and the assortment of apps it affords. As per the recent information by IDC (International Data Corporation), the

market-share of android smartphones is 83.8% till March' 2021 and it will grow to 85% till March'2025. Due to its widespread usage, we are deluged with a variety of smartphones apps that makes our life simple and easier. Flooding of these apps tempts attackers to design variety of malapp (malicious applications) directed toward smartphones to steal private information. Mobile devices including Smartphones generated about 54.8 % of worldwide web traffic [152] and analyzing this traffic leads to incredible results in detecting malapps. Analyzing traffic is accomplished by studying the patterns in the network traffic. Numerous traffic features were extracted from the network traffic patterns. Mainly Hypertext Transfer Protocol (HTTP) and Transmission Control Protocol (TCP) are two types of traffic that is prevalent in the smartphone ecosystem. Features extracted from HTTP and TCP are exploited in detecting the malapps. The HTTP header features could not detect the malapps in the encrypted traffic and TCP based detection models are impervious to encrypted traffic. Therefore, TCP flow-based detection methods are mainly exploited in detecting malapps.

Two widely used malware detection methods employed by researchers pivots around static and dynamic analysis. The amalgamation of these widely used detection methods is also exploited by some researcher's resulting in hybrid analysis. Most of the modern malwares depend on n/w interfaces to intercommunicate with attackers, therefore the n/w traffic based analysis technique is most suitable to detect android malapps. Static investigation based detection techniques failed to detect apps having code obfuscation, and conventional dynamic investigation based detection needs is quite cumbersome. N/w traffic based dynamic detection excerpts detection attributes from n/w traffic and uses Machine Learning (ML) techniques to categorize mobile apps. Our detection prototype is based on TCP based features.

In short, following are the main contributions:

1. We have proposed a traffic feature based fusion that comprises of optimal combination of multiple traffic features by cross diffusion of order and sparse graphs to produce a unified feature.
2. The unified feature vector generated is given to the three parallel ML classifiers and classifiers scores are fused to further enhance the accuracy attained by separate classifiers. Performance comparison with state-of-the-art methods were also performed.

5.2 Proposed Traffic Analysis Framework

The proposed traffic analysis framework for smartphone security analysis mainly comprised of mainly two stages i.e. feature unification and optimal fusion of the scores of three classifiers. Fig. 5.1 elucidates the overview of the suggested security analysis system. The suggested framework consists of four blocks namely, traffic feature extraction and vector formation, feature-fusion, classifier score-fusion, and a decision sub-block to accomplish efficient malapp detection. Extracted traffic features are converted into three traffic feature vectors. All the three traffic feature vectors are used for constructing similarity graphs by the means of cosine similarity. Similarity graphs are again converted to normalised graphs by using anchored normalization technique. Normalised graphs are again used to form the sparse and order graphs. These obtained sparse and order graphs are further cross diffused as to generate three fused traffic feature vectors. The three fused feature vectors are further concatenated to form the highly distinct unified feature vector. This distinct unified feature is discriminatory and given to three classifiers. The scores obtained from these classifiers are again optimally combined to classify a given test app.

In our method, we have used three classifiers viz. Random Forest (RF), k-Nearest Neighbour (KNN), and Support Vector Machine (SVM) for apps categorization. Also, we have leveraged modified PCR-5 rules for score fusion. Finally, in decision block score w_{PCR5} is matched with threshold w_{th} and particular test app is categorized into benign if $w_{PCR5} \geq w_{th}$ or otherwise malign.

Minutiae of suggested model is as follows:

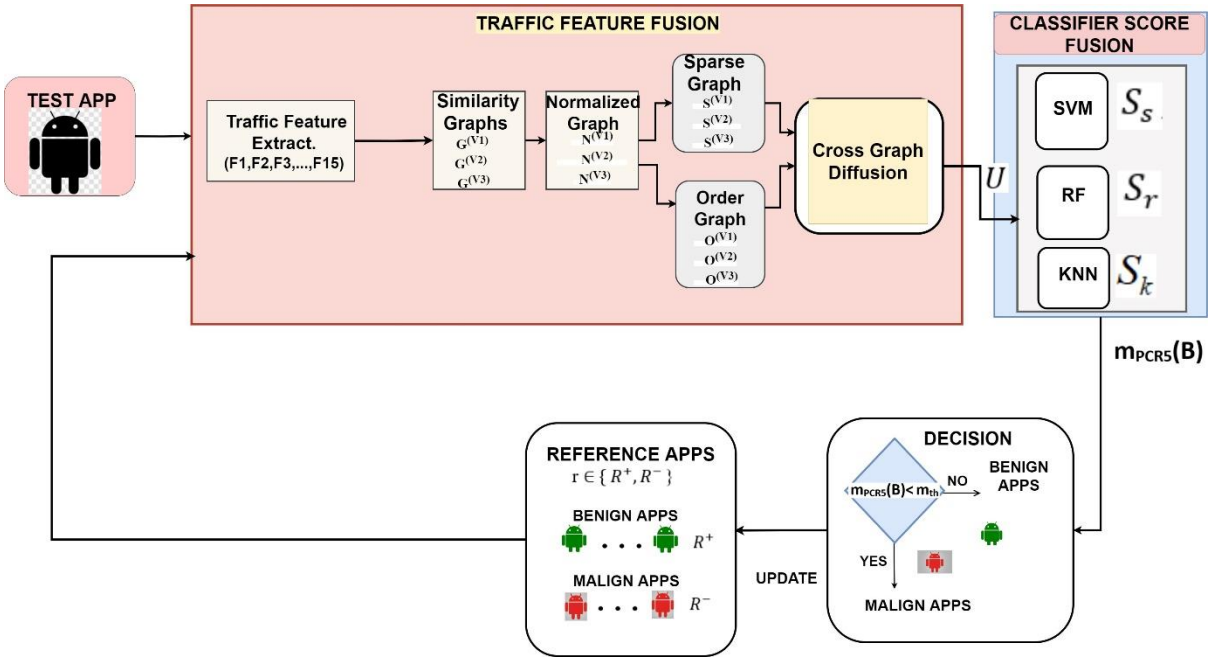

Figure 1. Proposed Traffic based framework for Smartphone Security Analysis.

Because of the numerous attributes encapsulated in the android based smartphone apps, individual ML algo shows its incompetence to categorize these apps accurately. Therefore, it is recommended to employ more than one ML in the detection of malapps. In the suggested method, we have employed three ML algo's viz. RF, SVM, and KNN for categorisation of apps. These three ML algo's are selected so as to improve the overall detection accuracy and offset the shortcomings of the single

classifiers. RF performs superbly when the dataset is large & it is not susceptible to outliers. SVM performs better in the limited dataset and it is optimal for binary classification. If there is no training period, then KNN performs best. It is simple, speedy and, time efficient. In the proposed framework, three classifiers supplement each other to improve the overall accuracy.

5.2.1 Traffic Gathering Platform

The traffic gathering platform is used to collect the malign and benign traffic data produced by malign and benign apps, respectively. A firewall is installed on the platform to guarantee its security. Fig. 5.2 shows our methodology for traffic gathering.

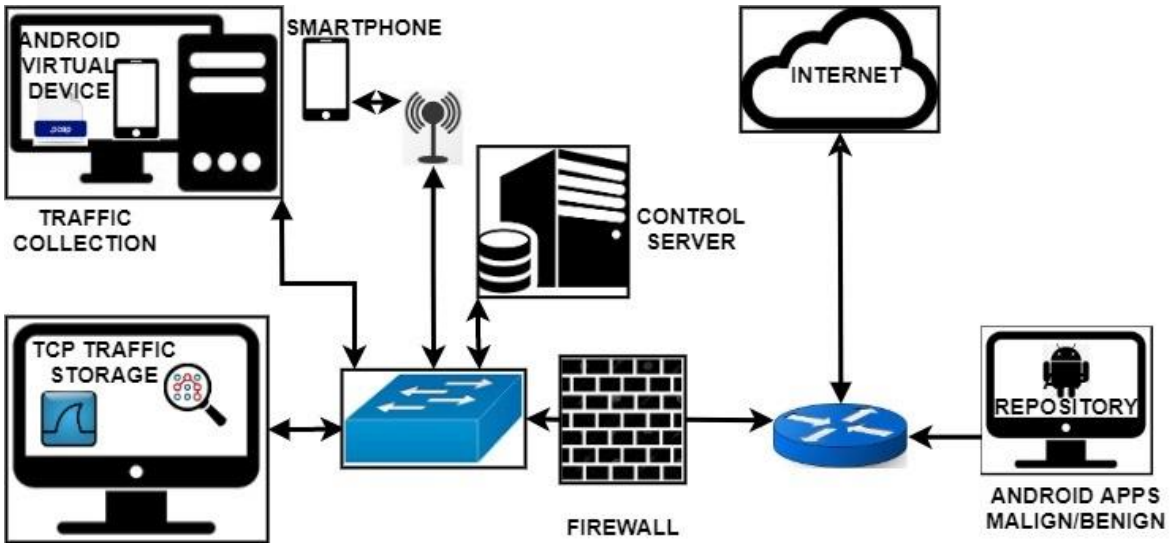

Figure 5.2. Traffic Gathering Platform

Table 5.1: Extracted TCP based Traffic Features

Feature Symbol	Feature Description
$F1$	Avg. no. of bytes sent
$F2$	Avg. no. of bytes received
$F3$	Total no. of headers bytes sent
$F4$	Avg. no. of bytes per second
$F5$	Ratio of the no. of incoming to outgoing bytes
$F6$	Avg. no. of packet sent per second
$F7$	Avg. no. of packet received per second
$F8$	Ratio of the no. of incoming to no. of outgoing packets
$F9$	Std. deviation of the packet- size sent
$F10$	Standard deviation of the packet- size received
$F11$	Avg. no. of packet sent per flow
$F12$	Avg. no. of packet received per flow
$F13$	Avg. no. of bytes sent per flow
$F14$	Avg. no. of bytes received per flow
$F15$	Std. deviation of the length of flow

The traffic gathering platform comprises four constituents, i.e., the control server, traffic collection module, app repository having downloaded malign and benign apps, and TCP traffic storage module containing only filtered TCP flows. These four components converse with the aid of LAN switch. The control server is controlling the traffic gathering job in the platform by assigning the job to the different modules. The apps from the apps repository are directed to the traffic collection module, where the android virtual machine (AVM) is used to run the apps and collect the corresponding traffic. The collected traffic is further directed to the TCP traffic storage module, where only TCP flows are stored and the rest of the traffic is filtered out. Here android emulators are used to install and run apps on AVM. AVM comprises of packed android s/w stack and it runs just like a physical smartphone. Apps are run on an emulator. Emulator is restarted to fuel the malign apps to generate malicious behaviour in the network traffic. A script in the python was written to extract the features from the TCP flows. The TCP features used are tabulated in Table 5.1.

5.2.2 Traffic Feature-Fusion

It comprises of three traffic vector creation after extracting the fifteen traffic features. These created traffic-feature vectors are again fused by optimal combination via non-linear cross-diffusion of created sparse and order graphs to draw complementary information and to create a highly discriminatory unified feature for classifying a test app.

5.2.2.1 Feature Extraction

Fifteen traffic features were extracted for a test app t along with N apps from reference dictionary, $d \in \{D^+, D^-\}, D^+$ comprises of benign apps and D^- comprises of malign app. Traffic feature extraction procedure shown in Fig 5.2. Reference dictionary apps are constantly renewed with most recent apps to improve the detection ability of the framework. The vectors formed are as follows:

$$V^1 = \{F1, F2, F3, F4, F5\} \quad (5.1)$$

$$V^2 = \{F6, F7, F8, F9, F10\} \quad (5.2)$$

$$V^3 = \{F11, F12, F13, F14, F15\} \quad (5.3)$$

We have built three feature vectors as stated in Eq. (5.1)-Eq. (5.3) for each test app and reference apps. In feature-fusion, traffic-features vectors for reference and test apps are utilized for graph formation. Test app's traffic-feature vectors represents one node and reference app's traffic-feature vector represents other node. Consequently, non-linear graphs are formed for all test app t corresponding to three traffic feature vector. For traffic feature vectors, V_t^1, V_t^2 and V_t^3 of test app t corresponding to three traffic-based features, we construct graphs $G^\phi = (Ver^\phi, E^\phi, w^\phi)$, where $\phi \in \{1, 2, 3\}$ corresponding to three traffic-based features and w^ϕ are edge weights acting as

similarity betwixt traffic-feature vectors of apps t and d where $d \in \{D^+, D^-\}$, Ver^{ϕ} correlates to the vertices of the created similarity graphs, E^{ϕ} correlates to the edges of the similarity graphs that characterize the association between test apps and the reference apps. $G^{\phi} \in \mathbb{R}^{n \times n}$ are constructed by cosine similarity between the three traffic-feature vectors of the test app t and reference apps d. For feature set values $\left(V_t^{\phi}, V_d^{\phi}\right)$, where $\phi \in \{1,2,3\}$ corresponds to three traffic-feature vector, the similarity edge weights are symbolized by vector $w^{\phi}(t,r)$, and is derived by the cosine similarity between the pair $\left(V_t^{\phi}, V_d^{\phi}\right)$ from the following Eq.(5.4)

$$w^{\phi}(t,d) = \frac{V_t^{\phi} * V_d^{\phi}}{\left\|V_t^{\phi}\right\| \left\|V_d^{\phi}\right\|} \tag{5.4}$$

Details of traffic-feature unification presented in subsequent subsection.

5.2.2.2 Feature Unification

Three traffic feature vectors created are fused in a way so as to extract complementary info in them by optimal non-linear cross-diffusion of generated sparse and order graphs. Features unification via cross-diffusion technique[19] was offered and its results confirm that the feature fusion by non-linear graphs techniques are significantly accurate than linear graph-centred approaches. Graph-built unification retains robust features of apps and rejects all the frail features that adds to misclassification.

Similarity graphs created using Eq.(4) for three traffic feature vectors are further normalized using an anchor $A^{\phi} = A_{avg}^{\phi} + A_{std}^{\phi}$, where A_{avg}^{ϕ} and A_{std}^{ϕ} are average and standard deviation of malicious score distribution for ϕ traffic feature vector. We consider only malicious scores for calculating the anchor. The normalized graphs N^{ϕ},

whose edge weights matrix $\bar{w}_j^{\phi}$, where $\forall j \in \{1,2,.....,n\}, \forall \phi \in \{1,2,3\}$ are constructed by Eq. (5.5)

$$\bar{w}_j^{\phi} = \begin{cases} \dfrac{w_j^{\phi} - \min(w^{\phi})}{2(A^{\phi} - \min(w^{\phi}))}, w_j^{\phi} \le A^{\phi} \\ 0.5 + \left(\dfrac{w_j^{\phi} - A^{\phi}}{\max(w_j^{\phi}) - A^{\phi}} \right), w_j^{\phi} > A^{\phi} \end{cases} \tag{5.5}$$

The obtained normalized graphs transformed to obtain the sparse S^{ϕ} and order O^{ϕ} graphs. Sparse graphs guarantee robustness to noise while boosting the strong info and suppressing the weak info corresponding to each traffic feature vector. Sparse graph, $S^{\phi} = \{Ver^{\phi}, E^{\phi}, \eta^{\phi}\}$ is built using KNN by Eq. (5.6)

$$\eta^{\phi} = \begin{cases} \bar{w}_j^{\phi}, \left(\bar{w}_j^{\phi} \in KNN_{N^{\phi}} \mid \nu \right) \\ 0, Otherwise \end{cases} \tag{5.6}$$

Where, ν is the parameter controlling the sparseness of graph. Edges corresponding to reference apps that are similar to the test apps are retained and the rest of the edges are removed to ensure robustness to noise. To discriminate the significant reference apps from the insignificant ones, each of the reference apps is assigned weight according to its order. Therefore, order(s) are assigned to each reference app on the basis of its similarity with test app, which is calculated by the edge weight between the test app and reference app in the normalized graph. The order graph $O^{\phi} = \{Ver^{\phi}, E^{\phi}, \mu^{\phi}\}$ having the weight matrix is constructed by Eq.(5.7) :

$$\mu^{\phi} = order(\bar{w}_j^{\phi}), \forall j \in \{1,2,...n\} \tag{5.7}$$

, where function *order*, allocates order(s) to each reference app.

The generated order and sparse graphs are further cross diffused to reinforce the strong connections between different traffic feature vectors and suppress the feeble connections. Order graphs averts biasness and sparse graphs guarantees elimination of any outlier behaviour. Cross diffusion [Eq.(9)] basically comprised of addition of sparse graphs of two other feature vector to form α^{ϕ^t} [Eq.(5.8)], where α^{ϕ^t} is the t^{th} element of the set ϕ and subsequently their multiplication with order graph of the feature vector with α^{ϕ^t} .

$$\alpha^{\phi^t} = \sum_{Y} \eta^{Y} \text{ where } Y \in \{\phi\} - \{\phi^t\} \tag{5.8}$$

$$\beta^{\phi^t} = \alpha^{\phi^t} \odot \mu^{\phi^t} \quad \forall \phi \in \{1,2,3\} \text{ , where } \odot \text{ denotes logical AND} \tag{5.9}$$

As can be seen in Fig 5.1, we add the sparse graph of two other feature vectors and then multiply it with the order graph of the feature vector turn by turn to generate three fused vectors. Eq.(5.9) can be further represented [Eq.(10,11,12)] in the form of fused vectors ($F^{V^1}, F^{V^2}, F^{V^3}$) , sparse vectors ($S^{V^1}, S^{V^2}, S^{V^3}$) and order vectors ($O^{V^1}, O^{V^2}, O^{V^3}$) as follows:

$$F^{V^1} = (S^{V^2} + S^{V^3}) \odot O^{V^1} \tag{5.10}$$

$$F^{V^2} = (S^{V^3} + S^{V^1}) \odot O^{V^2} \tag{5.11}$$

$$F^{V^3} = (S^{V^1} + S^{V^2}) \odot O^{V^3} \tag{5.12}$$

The above fused feature vectors F^{V^1}, F^{V^2} and F^{V^3} are concatenated to form the unified feature vector, U by Eq.(5.13)

$$U = F^{V^1} + F^{V^2} + F^{V^3} \tag{5.13}$$

This unified feature vector is given to three parallel classifier(s) and their scores are fused to classify apps. Details of Optimal classifier-fusion follows in the next sub-section.

5.2.3 Classifier Score-Fusion

Created U is fed to three classifiers connected in parallel. Obtained classification scores [S_s (SVM), S_r (RF), S_k (KNN)] are again fused by the classifier score fusion technique. There are various score fusion techniques reported in literature. Here we have chosen the PCR-5 [153]to solve the highly contradictory scores of the three classifiers. PCR-5 is used to solve ambiguous problems in multi-sensor score fusion. Android app detection is certainly ambiguous problem as we are uncertain whether the app is malicious or not. In the suggested model, three classifiers are selected and the fusion of the output of these classifiers can be modelled as multi-sensor score fusion problem as their outputs are independent of each other. Therefore, Android app detection satisfy all the conditions of PCR-5 theory. In our framework, frame of discernment Θ has two elements B and M corresponding to benign and malign. Classifier scores [S_s , S_r , S_k] are converted individual Basic Belief assignments or belief masses by the Eq.(5.14) below:

$$\begin{aligned} m_i(\mathrm{B}) &= C_i \times S_i(\mathrm{B}) \\ m_i(\mathrm{M}) &= 1 - C_i \times S_i(\mathrm{B}) \end{aligned} \qquad (5.14)$$

Where $i \in (s,r,k)$ & C_i denotes confidence measure of the single classifier.

These belief masses are combined by PCR-5 rules. Conjunctive consensus among the classifiers is assessed by Eq. (5.15) and Eq. (5.16)

$$m_{srk}(\mathrm{B}) = m_s(\mathrm{B}) * m_r(\mathrm{B}) * m_k(\mathrm{B}) \qquad (15)$$

$$m_{srk}(\mathrm{M}) = m_s(\mathrm{M}) * m_r(\mathrm{M}) * m_k(\mathrm{M}) \qquad (16)$$

Overall conflict among the classifiers is estimated by Eq. (5.17). It comprises of six partial conflicts-masses.

$$\begin{aligned} m_{srk}(\mathrm{B}\cap\mathrm{M}) = {} & m_s(\mathrm{B})*m_r(\mathrm{M})*m_k(\mathrm{M}) + m_r(\mathrm{B})*m_s(\mathrm{M})*m_k(\mathrm{M}) + \\ & m_k(\mathrm{B})*m_s(\mathrm{M})*m_r(\mathrm{M}) + m_s(\mathrm{M})*m_r(\mathrm{B})*m_k(\mathrm{B}) + \\ & m_r(\mathrm{M})*m_s(\mathrm{B})*m_k(\mathrm{B}) + m_k(\mathrm{M})*m_s(\mathrm{B})*m_r(\mathrm{B}) \end{aligned} \tag{5.17}$$

Six partial conflicts masses are further redistributed using PCR-5 rules in ratio to masses assisting to these partial-conflicts. The values of p_i and q_i are contribution to Benign and Malign masses following reallocation of partial conflicts, where i =1,...,6 and are determined by Eqs. (5.18-5.23)

$$\frac{p_1}{m_s(\mathrm{B})} = \frac{q_1}{m_r(\mathrm{M})*m_k(\mathrm{M})} = \frac{m_s(\mathrm{B})*m_r(\mathrm{M})*m_k(\mathrm{M})}{m_s(\mathrm{B})+m_r(\mathrm{M})*m_k(\mathrm{M})} \tag{18}$$

$$\frac{p_2}{m_r(\mathrm{B})} = \frac{q_2}{m_s(\mathrm{M})*m_k(\mathrm{M})} = \frac{m_r(\mathrm{B})*m_s(\mathrm{M})*m_k(\mathrm{M})}{m_r(\mathrm{B})+m_s(\mathrm{M})*m_k(\mathrm{M})} \tag{19}$$

$$\frac{p_3}{m_k(\mathrm{B})} = \frac{q_3}{m_s(\mathrm{M})*m_r(\mathrm{M})} = \frac{m_k(\mathrm{B})*m_s(\mathrm{M})*m_r(\mathrm{M})}{m_k(\mathrm{B})+m_s(\mathrm{M})*m_r(\mathrm{M})} \tag{20}$$

$$\frac{p_4}{m_r(\mathrm{B})*m_k(\mathrm{B})} = \frac{q_4}{m_s(\mathrm{M})} = \frac{m_s(\mathrm{M})*m_r(\mathrm{B})*m_k(\mathrm{B})}{m_s(\mathrm{M})+m_r(\mathrm{B})*m_k(\mathrm{B})} \tag{21}$$

$$\frac{p_5}{m_s(\mathrm{B})*m_k(\mathrm{B})} = \frac{q_5}{m_r(\mathrm{M})} = \frac{m_r(\mathrm{M})*m_s(\mathrm{B})*m_k(\mathrm{B})}{m_r(\mathrm{M})+m_s(\mathrm{B})*m_k(\mathrm{B})} \tag{22}$$

$$\frac{p_6}{m_s(\mathrm{B})*m_r(\mathrm{B})} = \frac{q_6}{m_k(\mathrm{M})} = \frac{m_k(\mathrm{M})*m_s(\mathrm{B})*m_r(\mathrm{B})}{m_k(\mathrm{M})+m_s(\mathrm{B})*m_r(\mathrm{B})} \tag{23}$$

Approximated contributions p_i and q_i where i=1,...,6 are add on to their respective conjunctive consensus. The final belief of the app being benign $m_{PCR5}(\mathrm{B})$ and that of app being malign $m_{PCR5}(\mathrm{M})$ are determined by Eq. (5.24) and Eq. (5.25) respectively.

$$m_{PCR5}(\mathrm{B}) = m_{srk}(\mathrm{B}) + \sum_{i=1}^{6} p_i \tag{5.24}$$

$$m_{PCR5}(\mathrm{M}) = m_{srk}(\mathrm{M}) + \sum_{i=1}^{6} q_i \tag{5.25}$$

The decision about the given test app t is taken after comparison of the $m_{PCR5}(\mathrm{B})$ with m_{th}. If $m_{PCR5}(\mathrm{B}) \geq m_{th}$, then the t is acknowledged as benign else it is taken as malign. Algo1 summarizes the pseudocode for proposed framework.

5.3 Experimental Validation

Quantitative analysis on various performance matrices like Precision, Accuracy, F1 Score, Specificity, and Sensitivity was performed on the suggested and two other state-of-the-art methods. Their ROC plots were also drawn for comparison. Following are the details of the evaluation process.

5.3.1 Databases

We have chosen about 3000 samples of malign apps and benign apps each downloaded from the benchmarked datasets. Using these apps and our traffic gathering platform, we have generated and captured the traffic filtering the TCP flows. We have selected only those apps from the datasets that produces the network traffic and filtered out the TCP based flows. Subsequently, traffic features as tabulated in

Table **5.1** were extracted from these flows using the script written in python. We have selected about 500 feature-flow vectors each corresponding to benign and malign flows from the total extracted flows corresponding to DB1, DB2, DB3, and DB4 datasets. Integrating these feature-flow vectors forms 2000 benign and malign feature-flows vector each. Further, we have applied ten-fold cross-validation procedure for training and testing done on five sets of feature-flow sets to reduce the biasness and variance and finally average values are taken as results. All investigations were done on MATLAB R2018a installed on 16GB RAM, i7, and 2.7 GHz processor.

Table **5.2**: Experimentation Dataset

App Type / Dataset	Malign Apps (M)	Benign Apps (B)	Source
DataBase(DB)1	750	750	Androzoo(M) [131] GooglePlay(B) [130]
DataBase(DB)2	750	750	AMD(M) [129] CICMalDroid2020(B) [132]
DataBase(DB)3	750	750	CICMalDroid2020(M) [132] GooglePlay(B) [130]
DataBase(DB)4	750	750	Drebin(M) [128] GooglePlay(B) [130]

5.3.2 Performance Assessment

Performance of the suggested technique was realized by the means of TCP features extracted from TCP flows corresponding to malign and benign apps as in Table 5.2 and calculating evaluation matrices through ten-fold cross validation. The proposed scheme was also compared with respect to training time and mean-time-to-detect

(MTTD) against three state-of-the-art approaches. Here, MTTD is the time taken to detect malapps. Evaluation matrices results are also compared with three state-of-the-arts techniques FED [154] , MMD [110] ,and LWN [111] . Detection Accuracy, F1 Score, Specificity, Sensitivity, and Precision are calculated using equations in Section 2.7, Chapter 2.

ROC curves for the proposed method and comparative methods FED, MMD, and LWN for flow sets corresponding to datasets DB1, DB2, DB3 and DB4 are depicted in Fig 5.3. The experimental results are presented in Table 5.3. It has been observed that the mean value of F score, detection accuracy, sensitivity, specificity and precision for the proposed framework are 0.98742, 98.74%, 99.04%, 98.44%, and 98.456% respectively. The highest value of the F score, detection accuracy, sensitivity, specificity and precision is 0.9890, 98.90%, 99.80%, 99.60, and 98.61 respectively.

Proposed technique outshines similar state-of-the-art traffic-based methods when evaluated on extracted TCP features as presented in Table 5.4. Enhancement for mean accuracy of suggested scheme by 6.28%, 4.34%, and 2.3% over LWN, FED, and MMD respectively have been achieved. This enhanced value of accuracy of the suggested framework was attained by optimal combination traffic features using cross diffusion strategy and fusion of classifier(s) score using DSmT based PCR-5 rule to form a clear-cut border for differentiating malign apps from benign apps. The suggested method was also compared with respect to MTTD of different state-of-the-art techniques. To calculate the MTTD of contemporary methods, learned models were fed with the 250 arbitrary apps for investigation. Our proposed method attains an amazing average analysis performance of 5.8 seconds per app. Similarly, the average analysis performance of LWN, FED, and MMD comes out to be 6.5 seconds, 7.3 seconds, and

6.9 seconds respectively. Hence, our proposed method outshined other methods in respect of detection time, detection accuracy, and efficacy in real-life apps scenarios. MTTD also confirms that suggested framework detects the apps with high accuracy in reasonable time.

The trained model is further tested on the network flows captured from the 20 different apps under unconstrained environment using the real smartphones instead of emulators and the performance comparison was done with our framework versus three different state-of-the-art methods viz. LWN , FED , and MMD and one self-proposed method RF+UF i.e. unified feature fed to the random forest algorithm. From Table 5.3, average accuracy obtained when the random apps are tested on LWN, FED, and MMD and RF+UF, and proposed method is 85.75%, 84.30%, 88.82%, 89.90%, and 95.10% respectively.

Performance comparison in terms of accuracy with existing methods using online captured data from 20 different apps chosen from the diverse sources are shown in Table 5.3. Here, we have taken Smart phone Samsung Galaxy S9 having android OS with 8 GB RAM and 64 GB storage for generating the network traffic. Our frame work detects the apps that generates the flows similar to one generated during the training phase with high accuracy.

Table **5.3.** Performance comparison in-terms-of accuracy with existing methods using captured data from 20 different apps.

Apps	UF+RF	LWN	FED	MMD	Proposed Method
	Accuracy				
AccuRadio	0.8900	0.8300	0.8500	0.8900	0.9400
Maps	0.8700	0.8100	0.8400	0.8100	0.9100
WhatsApp	0.8600	0.8200	0.8100	0.8700	0.9300
Outlook	0.8300	0.8100	0.8300	0.8500	0.9100
Mail	0.7800	0.8400	0.8500	0.8400	0.9300
Netflix	0.9000	0.7700	0.8900	0.9100	0.9500
Twitter	0.9300	0.7500	0.7600	0.9200	0.9400
Facebook	0.9100	0.9000	0.8700	0.9300	0.9500
Youtube	0.8800	0.8600	0.8800	0.9200	0.9700
Gmail	0.9200	0.9300	0.8900	0.9500	0.9600
DroidDream	0.8100	0.8400	0.9000	0.9200	0.9800
DroidKungFu1	0.8700	0.8800	0.9300	0.9400	0.9600
Buzz	0.8200	0.8400	0.9100	0.8900	0.9400
BlueScanner	0.8500	0.8400	0.9400	0.9300	0.9800
Plankton	0.8200	0.8500	0.9200	0.9100	0.9600
WallpaperGirls	0.8100	0.8200	0.8300	0.8600	0.9300
StylePhotoCollage	0.8700	0.8400	0.9100	0.9200	0.9400
PrivateSms	0.8300	0.8500	0.9400	0.8700	0.9900
PartMessage	0.8800	0.9000	0.9500	0.8900	0.9700
IdeaSecurity	0.8200	0.8800	0.9000	0.9600	0.9800

Table **5.4**. PM for Proposed methods and other comparative methods.

Dataset	Performance Metrics(PM)	LWN	FED	MMD	Proposed Method
DB1 FLOWSET	Accuracy	0.9250	0.9440	0.9620	0.9880
	Specificity	0.9100	0.9280	0.9540	0.9960
	Sensitivity	0.9400	0.9600	0.9700	0.9800
	F1 Score	0.9261	0.9449	0.9623	0.9879
	Precision	0.9126	0.9302	0.9547	0.9959
DB2 FLOWSET	Accuracy	0.9230	0.9380	0.9630	0.9890
	Specificity	0.9060	0.9360	0.9660	0.9860
	Sensitivity	0.9400	0.9400	0.9600	0.9920
	F1 Score	0.9243	0.9381	0.9629	0.9890
	Precision	0.9091	0.9363	0.9658	0.9861
DB3 FLOWSET	Accuracy	0.9260	0.9440	0.9630	0.9881
	Specificity	0.9140	0.9420	0.9460	0.9780
	Sensitivity	0.9380	0.9460	0.9800	0.9980
	F1 Score	0.9269	0.9441	0.9636	0.9881
	Precision	0.9160	0.9422	0.9478	0.9784
DB4 FLOWSET	Accuracy	0.9220	0.9460	0.9660	0.9870
	Specificity	0.8860	0.9260	0.9560	0.9780
	Sensitivity	0.9580	0.9660	0.9760	0.9960
	F1 Score	0.9247	0.9471	0.9663	0.9871
	Precision	0.8937	0.9288	0.9569	0.9784
INTEGRATED FLOWSET	Accuracy	0.9270	0.9480	0.9680	0.9850
	Specificity	0.9405	0.9455	0.9645	0.9840
	Sensitivity	0.9133	0.9505	0.9715	0.9860
	F1 Score	0.9260	0.9481	0.9681	0.9850
	Precision	0.9388	0.9458	0.9647	0.9840

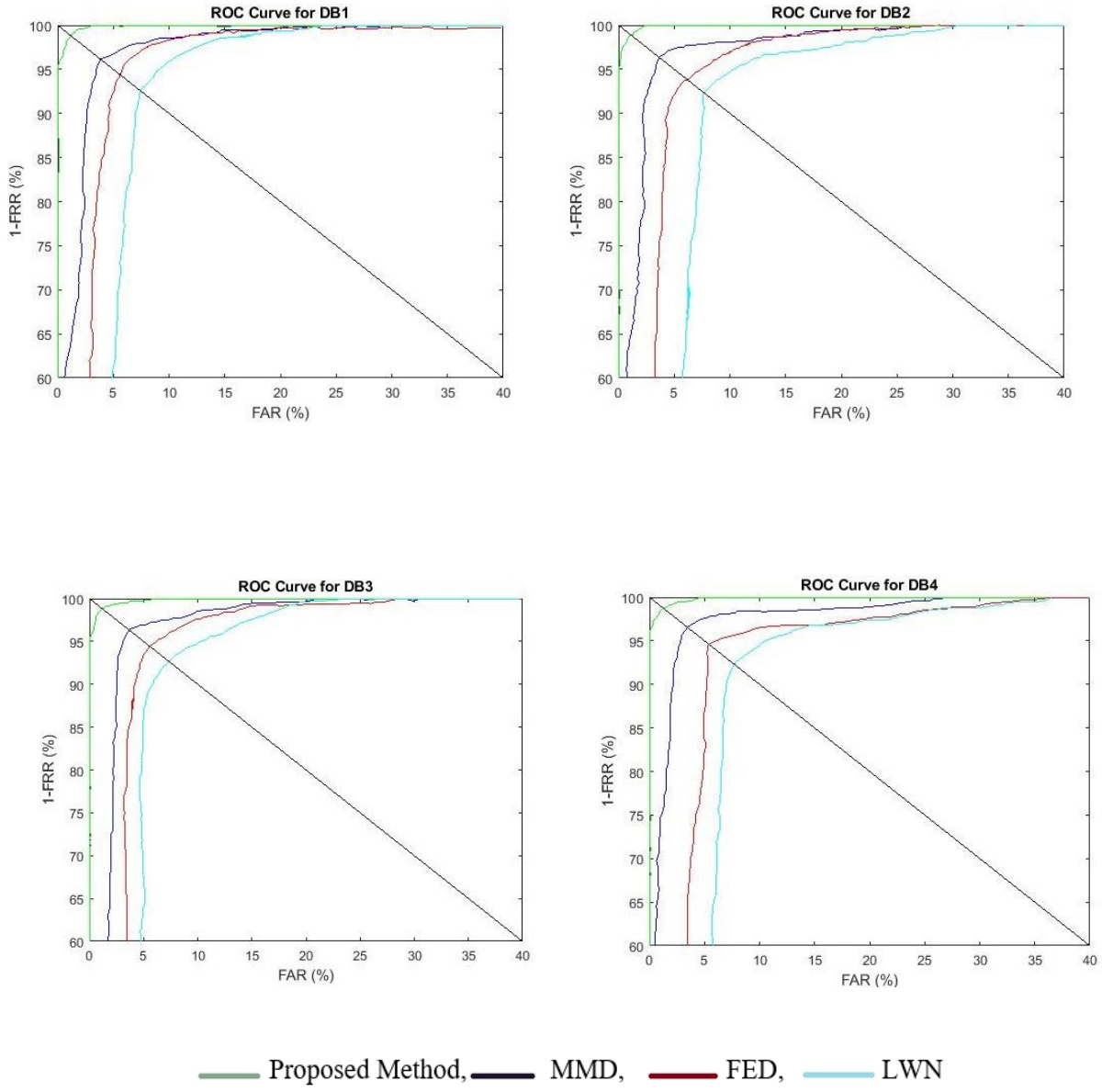

Figure **5.3:** ROC curves comparison for the proposed method and comparative methods MMD, FED, LWN for flow sets corresponding to datasets DB1, DB2, DB3 and DB4.

5.4 Conclusion

The significant highlights of this research work are as follows:

- A novel traffic analysis framework is presented wherein multiple TCP-based traffic features were optimally combined by diffusing the features by graph based cross diffusion leading to the formation of a unified feature vector.

- It has been observed that the mean value of F score, detection accuracy, sensitivity, specificity and precision for the proposed framework are 0.98742, 98.74%, 99.04%, 98.44%, and 98.456% respectively.
- The highest value of the F score, detection accuracy, sensitivity, specificity and precision is 0.9890, 98.90%, 99.80%, 99.60, and 98.61 respectively.
- Proposed technique outshines similar state-of-the-art traffic-based methods when evaluated on extracted TCP features as presented in Table 5.4. Enhancement for mean accuracy of suggested scheme by 6.28%, 4.34%, and 2.3% when compared with LWN, FED, and MMD methods respectively have been achieved. This enhanced value of accuracy of the suggested framework was attained by optimal combination traffic features using cross diffusion strategy and fusion of classifier(s) score using DSmT based PCR-5 rule to form a clear-cut border for differentiating malign apps from benign apps.
- The suggested method was also compared with respect to MTTD of different state-of-the-art techniques. To calculate the MTTD of contemporary methods, learned models were fed with the 250 arbitrary apps for investigation. Our proposed method attains an amazing average analysis performance of 5.8 seconds per app. Similarly, the average analysis performance of LWN, FED, and MMD comes out to be 6.5 seconds per app, 7.3 seconds per app, and 6.9 seconds per app respectively. Hence, our proposed method outshined other methods in respect of detection time, detection accuracy, and efficacy in real-life apps scenarios. MTTD also confirms that suggested framework detects the apps with high accuracy in reasonable time.

- Average accuracy obtained when the random apps are tested on LWN, FED, and MMD and RF+UF, and proposed method is 85.75%, 84.30%, 88.82%, 89.90%, and 95.10% respectively.

The experimental results along with other findings were published in [155].

.

Chapter 6

Conclusions & Future Directions

This chapter will summarize the major contributions and achievements that come out of the present work. Despite the significant contributions, no research is said to be complete unless it directs to a few topics for future research. Hence, the potential work that can be explored further is briefly discussed as directions to future work in the Section. 6.2. The summary of the major contributions follows in the coming subsection.

6.1 Summary of Major Contributions

The essence of this thesis work was to design and develop efficient techniques for smartphone security analysis. In order to address the limitations of various aspects in this field, several innovative methods have been suggested under current work which are summarized as follows:

- Challenges faced in the identification of malapps in the smartphones motivated us to develop an efficient multi-fusion based android malapp detection method based on static analysis. In our methodology, eight static features are exploited for development of solution. Our multi-fusion technique is a two stage fusion approach. In first stage, deduced static features are fused into a single unified robust vector through extraction of the complementary information from eight features using non-linear graph unification. In second stage, optimal classifiers

are used for classification of an app. Proposed framework has been designed to make classification robust to noise and hence help in drastically improving the score distribution. Classification is performed by training RF, SVM, and NB classifiers followed by classifier scores fusion using PCR-6 rule that resolved conflicts amongst classifiers besides redistributing the conflicts efficiently. Qualitative investigations of outcomes disclosed that proposed optimal classifier broadened the score-distribution of malign and benign apps. Moreover, our method has attained average accuracy of 98.97%, average equal error rate of 1.04%, average F1 score value of 0.9936 and average sensitivity value of 0.9905 when evaluated over benchmarked datasets. Quantitative analysis of suggested method vs. state-of-the-art techniques reveal that proposed method outperforms all of them.

- To tackle the challenges of effective detection of ever-evolving android malapps, we suggested a novel hybrid malapp detection scheme. Here, the optimal combination of static and dynamic features by cross-diffusion followed by fuzzy-based score level fusion was proposed. In the suggested framework, we have used five static and two dynamic features to form static and dynamic feature vectors. These feature vectors are further fused after the formation of normalized and refined graphs through the non-linear graph diffusion method. The fused feature vector is then given to an optimal classifier comprising of RF and SVM classifiers. A remarkable benefit of our method is that it can extract the static and dynamic features in each app almost in real-time. In sum, the unification of static and dynamic feature achieve highly distinct features. Adoption of fuzzy-based fusion of classifier scores not only create clear boundary but also achieve optimal performance. Our technique has accomplished mean value of accuracy, specificity, sensitivity, and F1 score as

98.62%, 98.634%, 98.604 %, and 99.16% respectively when estimated on datasets. Experimental results reveal that our technique outstrips other state-of-the-art methods.

- The precipitous increase in the number of android malicious apps drive mobile-users to take extra security precautions, and makes the malapp detection a significant challenge. With the aim of detecting the android malapps, we suggested an android traffic based framework to detect android malapps. In our framework, fifteen TCP-based features were extracted to build the solution. Extracted features were optimally combined by diffusing the features by graph based cross diffusion leading to the formation of a unified feature vector. The unified feature vector was again given to three classifiers viz. SVM, RF, and KNN. The classifiers scores obtained were further fused by PCR-5 rules. Our method has accomplished the average detection accuracy, average F1 score, average precision, and average sensitivity of when assessed on the traffic features extracted from the flows corresponding to benchmarked dataset apps. Comparison of the evaluation matrices of proposed framework with other state-of-the-art approaches shows that suggested framework is better. Our system can also categorize the encrypted traffic successfully.

6.2 Future Directions

In this thesis, numerous smartphone security analysis frameworks were investigated and explored in detail to provide novel contribution in this area. But there are some research dimensions that arise out of the current work which demand future study.

These dimensions are summarized as directions to future work and are enlisted as follows:

- The main reason for misclassification is non-inclusion of some dynamic features and other structure related static features in our model. Therefore, we will extend our framework to improve the detection efficiency by including these features. Also, training on more datasets covering the different families of malware from diverse sources will solve under-fitting and overfitting problems in detection.
- HTTPS-based dynamic features in addition to TCP features are to be incorporated to improve detection accuracy. Furthermore, training on more datasets comprising diverse malware families will further add robustness to the framework.
- Other popular smartphone platform like iOS must explored in future.
- Investigations on smart devices other than smartphones like smart cars, smart thermostats, smart locks, smart refrigerators, smart watches, smart bands, smart key chains etc. and other miniature devices like sensors etc. prone to malwares may be taken up in future. Also, 5G and 6G compatible smart devices must be taken up in the future.
- Many ML algorithm are susceptible to spoofing, poisoning, inversion, and impersonate attacks. The countermeasures for these attacks may be incorporated in the future.
- The new-fangled malwares deteriorates the detection capabilities of the framework. Future framework will be developed to improve the zero day identification ability. Also, framework scalability on large datasets of apps

should be verified. Tools handling the big data with fast processing should be utilised.

- Advanced ML techniques like deep learning technology must be exploited in the frameworks. Reinforcement learning, a form of deep learning shows incredible results in dynamic frameworks to segregate the malwares. Transferred learning is another technique to cater the problem of limited samples in families. Crowdsourcing should be employed to solve malware family detection issues.

- This study further demands the possibility of analysis on other benchmarked databases and execution of other performance evaluation metrics.

References

[1] S. Wang, Q. Yan, Z. Chen, B. Yang, C. Zhao, and M. Conti, "Detecting Android Malware Leveraging Text Semantics of Network Flows," *IEEE Trans. Inf. Forensics Secur.*, vol. 13, no. 5, pp. 1096–1109, May 2018, doi: 10.1109/TIFS.2017.2771228.

[2] "IDC, Smartphone Market Share, 2020," 2020. https://www.idc.com/promo/smartphone-market-share/os (accessed Feb. 20, 2021).

[3] P. Faruki *et al.*, "Android security: A survey of issues, malware penetration, and defenses," *IEEE Commun. Surv. Tutorials*, vol. 17, no. 2, pp. 998–1022, 2015, doi: 10.1109/COMST.2014.2386139.

[4] S. Rastogi, K. Bhushan, and B. B. Gupta, "Android Applications Repackaging Detection Techniques for Smartphone Devices," *Phys. Procedia*, vol. 78, no. December 2015, pp. 26–32, 2016, doi: 10.1016/j.procs.2016.02.006.

[5] Q. Han, V. S. Subrahmanian, and Y. Xiong, "Android Malware Detection via (Somewhat) Robust Irreversible Feature Transformations," *IEEE Trans. Inf. Forensics Secur.*, vol. 15, no. iii, pp. 3511–3525, 2020, doi: 10.1109/TIFS.2020.2975932.

[6] H. Zhang, S. Luo, Y. Zhang, and L. Pan, "An Efficient Android Malware Detection System Based on Method-Level Behavioral Semantic Analysis," *IEEE Access*, vol. 7, pp. 69246–69256, *20*19, doi: 10.1109/ACCESS.2019.2919796.

[7] H. Hasan, B. Tork Ladani, and B. Zamani, "MEGDroid: A model-driven event generation framework for dynamic android malware analysis," *Inf. Softw. Technol.*, vol. 135, pp. 1–23, 2021, doi: 10.1016/j.infsof.2021.106569.

[8] M. Dhalaria and E. Gandotra, "A Hybrid Approach for Android Malware Detection and Family Classification," *Int. J. Interact. Multimed. Artif. Intell.*, vol. In Press, no. In Press, p. 1, 2020, doi: 10.9781/ijimai.2020.09.001.

[9] A. Arora, S. Garg, and S. K. Peddoju, "Malware detection using network traffic analysis in android based mobile devices," *Proc. - 2014 8th Int. Conf. Next Gener. Mob. Appl. Serv. Technol. NGMAST 2014*, pp. 66–71, 2014, doi: 10.1109/NGMAST.2014.57.

[10] M. McLaughlin, ""Android Versions Guide: Everything You Need to Know"."

https://www.lifewire.com/android-versions-4173277.

[11] J. Gao, L. Li, P. Kong, T. F. Bissyande, and J. Klein, "Understanding the Evolution of Android App Vulnerabilities," *IEEE Trans. Reliab.*, vol. 70, no. 1, pp. 212–230, 2021, doi: 10.1109/TR.2019.2956690.

[12] T. Alladi, V. Chamola, B. Sikdar, and K. K. R. Choo, "Consumer IoT: Security Vulnerability Case Studies and Solutions," *IEEE Consum. Electron. Mag.*, vol. 9, no. 2, pp. 17–25, 2020, doi: 10.1109/MCE.2019.2953740.

[13] R. Afzal and R. K. Murugesan, "Implementation of a Malicious Traffic Filter Using Snort and Wireshark as a Proof of Concept to Enhance Mobile Network Security," 2022.

[14] R. M. Abdullah, A. Z. Abualkishik, N. M. Isaacc, A. A. Alwan, and Y. Gulzar, "An investigation study for risk calculation of security vulnerabilities on android applications," *Indones. J. Electr. Eng. Comput. Sci.*, vol. 25, no. 3, pp. 1736–1748, 2022, doi: 10.11591/ijeecs.v25.i3.pp1736-1748.

[15] B. Yang, R. Chen, K. Huang, J. Yang, and W. Gao, "Eavesdropping User Credentials via GPU Side Channels on Smartphones," in *International Conference on Architectural Support for Programming Languages and Operating Systems - ASPLOS*, 2022, pp. 285–299, doi: 10.1145/3503222.3507757.

[16] K. Tian, D. Yao, B. G. Ryder, G. Tan, and G. Peng, "Detection of Repackaged Android Malware with Code-Heterogeneity Features," *IEEE Trans. Dependable Secur. Comput.*, vol. 17, no. 1, pp. 64–77, 2020, doi: 10.1109/TDSC.2017.2745575.

[17] W. Wang *et al.*, "Constructing Features for Detecting Android Malicious Applications: Issues, Taxonomy and Directions," *IEEE Access*, vol. 7, pp. 67602–67631, 2019, doi: 10.1109/ACCESS.2019.2918139.

[18] M. Fan, J. Liu, W. Wang, H. Li, Z. Tian, and T. Liu, "DAPASA: Detecting Android Piggybacked Apps Through Sensitive Subgraph Analysis," *IEEE Trans. Inf. Forensics Secur.*, vol. 12, no. 8, pp. 1772–1785, 2017, doi: 10.1109/TIFS.2017.2687880.

[19] H. J. Zhu, T. H. Jiang, B. Ma, Z. H. You, W. L. Shi, and L. Cheng, "HEMD: a highly efficient random forest-based malware detection framework for Android," *Neural Comput. Appl.*, vol. 30, no. 11, pp. 3353–3361, Dec. 2018, doi: 10.1007/s00521-017-2914-y.

[20] X. Wang, D. Zhang, X. Su, and W. Li, "Mlifdect: Android malware detection based on parallel machine learning and information fusion," *Secur. Commun. Networks*, vol. 2017, 2017, doi: 10.1155/2017/6451260.

[21] L. Cen, C. S. Gates, L. Si, and N. Li, “A Probabilistic Discriminative Model for Android Malware Detection with Decompiled Source Code,” *IEEE Trans. Dependable Secur. Comput.*, vol. 12, no. 4, pp. 400–412, 2015, doi: 10.1109/TDSC.2014.2355839.

[22] G. Tao, Z. Zheng, S. Member, Z. Guo, and M. R. Lyu, “MalPat : Mining Patterns of Malicious and Benign,” *IEEE Trans. Reliab.*, pp. 1–15, 2017, doi: 10.1109/TR.2017.2778147.

[23] Z. Ma, H. Ge, Y. Liu, M. Zhao, and J. Ma, “A Combination Method for Android Malware Detection Based on Control Flow Graphs and Machine Learning Algorithms,” *IEEE Access*, vol. 7, no. c, pp. 21235–21245, 2019, doi: 10.1109/ACCESS.2019.2896003.

[24] J. Li, L. Sun, Q. Yan, Z. Li, W. Srisa-An, and H. Ye, “Significant Permission Identification for Machine-Learning-Based Android Malware Detection,” *IEEE Trans. Ind. Informatics*, vol. 14, no. 7, pp. 3216–3225, 2018, doi: 10.1109/TII.2017.2789219.

[25] N. Milosevic, A. Dehghantanha, and K. K. R. Choo, “Machine learning aided Android malware classification,” *Comput. Electr. Eng.*, vol. 61, pp. 266–274, 2017, doi: 10.1016/j.compeleceng.2017.02.013.

[26] H. Kang, J. W. Jang, A. Mohaisen, and H. K. Kim, “Detecting and classifying android malware using static analysis along with creator information,” *Int. J. Distrib. Sens. Networks*, vol. 2015, 2015, doi: 10.1155/2015/479174.

[27] A. Narayanan, S. Member, M. Chandramohan, and S. Member, “Context-Aware , Adaptive , and Scalable Android,” *IEEE Trans. Emerg. Top. Comput. Intell.*, vol. 1, no. 3, pp. 157–175, 2017.

[28] W. Wang, Z. Gao, M. Zhao, Y. Li, J. Liu, and X. Zhang, “DroidEnsemble: Detecting Android Malicious Applications with Ensemble of String and Structural Static Features,” *IEEE Access*, vol. 6, no. c, pp. 31798–31807, 2018, doi: 10.1109/ACCESS.2018.2835654.

[29] T. Kim, B. Kang, M. Rho, S. Sezer, and E. G. Im, “A multimodal deep learning method for android malware detection using various features,” *IEEE Trans. Inf. Forensics Secur.*, vol. 14, no. 3, pp. 773–788, 2019, doi: 10.1109/TIFS.2018.2866319.

[30] M. Fan *et al.*, “Android malware familial classification and representative sample selection via frequent subgraph analysis,” *IEEE Trans. Inf. Forensics Secur.*, vol. 13, no. 8, pp. 1890–1905, 2018, doi: 10.1109/TIFS.2018.2806891.

[31] S. Y. Yerima and S. Sezer, “DroidFusion: A Novel Multilevel Classifier Fusion Approach for Android Malware Detection,” *IEEE Trans. Cybern.*, vol. 49, no. 2, pp. 453–466, 2019, doi: 10.1109/TCYB.2017.2777960Y.

[32] A. Arora, S. K. Peddoju, and M. Conti, “PermPair: Android Malware Detection Using Permission Pairs,” *IEEE Trans. Inf. Forensics Secur.*, vol. 15, no. 8, pp. 1968–1982, 2020, doi: 10.1109/TIFS.2019.2950134.

[33] W. Wang, Y. Li, X. Wang, J. Liu, and X. Zhang, “Detecting Android malicious apps and categorizing benign apps with ensemble of classifiers,” *Futur. Gener. Comput. Syst.*, vol. 78, pp. 987–994, 2018, doi: 10.1016/j.future.2017.01.019.

[34] J. Song, C. Han, K. Wang, J. Zhao, R. Ranjan, and L. Wang, “An integrated static detection and analysis framework for android,” *Pervasive Mob. Comput.*, vol. 32, pp. 15–25, 2016, doi: 10.1016/j.pmcj.2016.03.003.

[35] J. Qiu *et al.*, “A3CM: Automatic Capability Annotation for Android Malware,” *IEEE Access*, vol. 7, pp. 147156–147168, 2019, doi: 10.1109/ACCESS.2019.2946392.

[36] O. Olukoya, L. Mackenzie, and I. Omoronyia, “Security-oriented view of app behaviour using textual descriptions and user-granted permission requests,” *Comput. Secur.*, vol. 89, p. 101685, 2020, doi: 10.1016/j.cose.2019.101685.

[37] S. Alam, S. A. Alharbi, and S. Yildirim, “Mining nested flow of dominant APIs for detecting android malware,” *Comput. Networks*, vol. 167, p. 107026, 2020, doi: 10.1016/j.comnet.2019.107026.

[38] R. Mateless, D. Rejabek, O. Margalit, and R. Moskovitch, “Decompiled APK based malicious code classification,” *Futur. Gener. Comput. Syst.*, vol. 110, pp. 135–147, 2020, doi: 10.1016/j.future.2020.03.052.

[39] H. J. Zhu, Z. H. You, Z. X. Zhu, W. L. Shi, X. Chen, and L. Cheng, “DroidDet: Effective and robust detection of android malware using static analysis along with rotation forest model,” *Neurocomputing*, vol. 272, pp. 638–646, 2018, doi: 10.1016/j.neucom.2017.07.030.

[40] A. Alotaibi, “Identifying Malicious Software Using Deep Residual Long-Short Term Memory,” *IEEE Access*, vol. 7, pp. 163128–163137, 2019, doi: 10.1109/ACCESS.2019.2951751.

[41] K. Xu, Y. Li, and R. H. Deng, “ICCDetector: ICC-Based Malware Detection on Android,” *IEEE Trans. Inf. Forensics Secur.*, vol. 11, no. 6, pp. 1252–1264, 2016, doi: 10.1109/TIFS.2016.2523912.

[42] H. Peng *et al.*, “Using probabilistic generative models for ranking risks of Android apps,” *Proc. ACM Conf. Comput. Commun. Secur.*, pp. 241–252, 2012, doi: 10.1145/2382196.2382224.

[43] Y. Zhou, X. Jiang, and A. Nazish, “Dissecting Android Malware : Characterization and Evolution

Summarized by : Nazish Asad," *Proc. - IEEE Symp. Secur. Priv.*, no. 4, pp. 95–109, 2011.

[44] Y. Aafer, W. Du, and H. Yin, "DroidAPIMiner: Mining API-level features for robust malware detection in android," *Lect. Notes Inst. Comput. Sci. Soc. Telecommun. Eng. LNICST*, vol. 127 LNICST, pp. 86–103, 2013, doi: 10.1007/978-3-319-04283-1_6.

[45] "VIRUS SHARE 2020." [Online]. Available: https://virusshare.com.

[46] "Anzhi market 2020." [Online]. Available: http://www.anzhi.com.

[47] "Lenovomm market 2020." [Online]. Available: http://www.lenovomm.com.

[48] "Wandoujia market 2020." [Online]. Available: http://www.wandoujia.com.

[49] H. Cai, N. Meng, B. Ryder, and D. Yao, "DroidCat: Effective android malware detection and categorization via app-level profiling," *IEEE Trans. Inf. Forensics Secur.*, vol. 14, no. 6, pp. 1455–1470, 2019, doi: 10.1109/TIFS.2018.2879302.

[50] A. Saracino, D. Sgandurra, G. Dini, and F. Martinelli, "MADAM: Effective and Efficient Behavior-based Android Malware Detection and Prevention," *IEEE Trans. Dependable Secur. Comput.*, vol. 15, no. 1, pp. 83–97, 2018, doi: 10.1109/TDSC.2016.2536605.

[51] P. Feng, J. Ma, C. Sun, X. Xu, and Y. Ma, "A novel dynamic android malware detection system with ensemble learning," *IEEE Access*, vol. 6, no. c, pp. 30996–31011, 2018, doi: 10.1109/ACCESS.2018.2844349.

[52] A. Martín, V. Rodríguez-Fernández, and D. Camacho, "CANDYMAN: Classifying Android malware families by modelling dynamic traces with Markov chains," *Eng. Appl. Artif. Intell.*, vol. 74, no. July 2017, pp. 121–133, 2018, doi: 10.1016/j.engappai.2018.06.006.

[53] T. Bhatia and R. Kaushal, "Malware detection in android based on dynamic analysis," *2017 Int. Conf. Cyber Secur. Prot. Digit. Serv. Cyber Secur. 2017*, 2017, doi: 10.1109/CyberSecPODS.2017.8074847.

[54] L. Singh and M. Hofmann, "Dynamic behavior analysis of android applications for malware detection," *ICCT 2017 - Int. Conf. Intell. Commun. Comput. Tech.*, vol. 2018-Janua, no. 2013, pp. 1–7, 2018, doi: 10.1109/INTELCCT.2017.8324010.

[55] S. Zhang and X. Xiao, "CSCdroid: Accurately Detect Android Malware via Contribution-Level-Based System Call Categorization," *Proc. - 16th IEEE Int. Conf. Trust. Secur. Priv. Comput. Commun. 11th IEEE Int. Conf. Big Data Sci. Eng. 14th IEEE Int. Conf. Embed. Softw. Syst.*, no. 1, pp. 193–200, 2017, doi: 10.1109/Trustcom/BigDataSE/ICESS.2017.237.

[56] S. D. Yalew, G. Q. Maguire, S. Haridi, and M. Correia, "T2Droid: A trustzone-based dynamic

analyser for android applications," *Proc. - 16th IEEE Int. Conf. Trust. Secur. Priv. Comput. Commun. 11th IEEE Int. Conf. Big Data Sci. Eng. 14th IEEE Int. Conf. Embed. Softw. Syst.*, no. Vm, pp. 240–247, 2017, doi: 10.1109/Trustcom/BigDataSE/ICESS.2017.243.

[57] S. Iqbal and M. Zulkernine, "SpyDroid: A Framework for Employing Multiple Real-Time Malware Detectors on Android," *MALWARE 2018 - Proc. 2018 13th Int. Conf. Malicious Unwanted Softw.*, pp. 33–40, 2019, doi: 10.1109/MALWARE.2018.8659365.

[58] M. Jaiswal, Y. Malik, and F. Jaafar, "Android gaming malware detection using system call analysis," *6th Int. Symp. Digit. Forensic Secur. ISDFS 2018 - Proceeding*, vol. 2018-Janua, pp. 1–5, 2018, doi: 10.1109/ISDFS.2018.8355360.

[59] Z. U. Rehman *et al.*, "Machine learning-assisted signature and heuristic-based detection of malwares in Android devices," *Comput. Electr. Eng.*, vol. 69, pp. 828–841, 2018, doi: 10.1016/j.compeleceng.2017.11.028.

[60] S. Arshad, M. A. Shah, A. Wahid, A. Mehmood, H. Song, and H. Yu, "SAMADroid: A Novel 3-Level Hybrid Malware Detection Model for Android Operating System," *IEEE Access*, vol. 6, pp. 4321–4339, 2018, doi: 10.1109/ACCESS.2018.2792941.

[61] W. Zhang, H. Wang, H. He, and P. Liu, "DAMBA: Detecting Android Malware by ORGB Analysis," *IEEE Trans. Reliab.*, vol. 69, no. 1, pp. 55–69, 2020, doi: 10.1109/TR.2019.2924677.

[62] A. T. Kabakus and I. A. Dogru, "An in-depth analysis of Android malware using hybrid techniques," *Digit. Investig.*, vol. 24, pp. 25–33, 2018, doi: 10.1016/j.diin.2018.01.001.

[63] A. Guerra-Manzanares, H. Bahsi, and S. Nõmm, "KronoDroid: Time-based hybrid-featured dataset for effective android malware detection and characterization," *Comput. Secur.*, vol. 110, p. 102399, 2021, doi: 10.1016/j.cose.2021.102399.

[64] X. Wang, Y. Yang, and S. Zhu, "Automated Hybrid Analysis of Android Malware through Augmenting Fuzzing with Forced Execution," *IEEE Trans. Mob. Comput.*, vol. 18, no. 12, pp. 2768–2782, 2019, doi: 10.1109/TMC.2018.2886881.

[65] S. Arshad, M. A. Shah, A. Wahid, A. Mehmood, H. Song, and H. Yu, "SAMADroid: A Novel 3-Level Hybrid Malware Detection Model for Android Operating System," *IEEE Access*, vol. 6, pp. 4321–4339, Jan. 2018, doi: 10.1109/ACCESS.2018.2792941.

[66] W. C. Kuo, T. P. Liu, and C. C. Wang, "Study on android hybrid malware detection based on machine learning," *2019 IEEE 4th Int. Conf. Comput. Commun. Syst. ICCCS 2019*, pp. 31–35, 2019, doi: 10.1109/CCOMS.2019.8821665.

[67] S. J. Hussain, U. Ahmed, H. Liaquat, S. Mir, N. Z. Jhanjhi, and M. Humayun, "IMIAD: Intelligent malware identification for android platform," *2019 Int. Conf. Comput. Inf. Sci. ICCIS 2019*, pp. 1–6, 2019, doi: 10.1109/ICCISci.2019.8716471.

[68] D. C. Alejandro Martín, Raúl Lara-Cabrera, "Android malware detection through hybrid features fusion and ensemble classifiers: the AndroPyTool framework and the OmniDroid dataset," *Inf. Fusion*, vol. 52, no. 2019, pp. 128–142, 2020, doi: https://doi.org/10.1016/j.inffus.2018.12.006.

[69] Q. Fang, X. Yang, and C. Ji, "A hybrid detection method for android malware," *Proc. 2019 IEEE 3rd Inf. Technol. Networking, Electron. Autom. Control Conf. ITNEC 2019*, no. Itnec, pp. 2127–2132, 2019, doi: 10.1109/ITNEC.2019.8729017.

[70] C. Ding, N. Luktarhan, B. Lu, and W. Zhang, "A hybrid analysis-based approach to android malware family classification," *Entropy*, vol. 23, no. 8, 2021, doi: 10.3390/e23081009.

[71] A. T. KABAKUŞ, "Hybroid: A Novel Hybrid Android Malware Detection Framework," *Erzincan Üniversitesi Fen Bilim. Enstitüsü Derg.*, vol. 14, no. 1, pp. 331–356, 2021, doi: 10.18185/erzifbed.806683.

[72] A. Karim, V. Chang, and A. Firdaus, "Android botnets: A proof-of-concept using hybrid analysis approach," *J. Organ. End User Comput.*, vol. 32, no. 3, pp. 50–67, 2020, doi: 10.4018/JOEUC.2020070105.

[73] M. Conti, Q. Q. Li, A. Maragno, and R. Spolaor, "The dark side(-Channel) of Mobile Devices: A survey on network traffic analysis," *IEEE Commun. Surv. Tutorials*, vol. 20, no. 4, pp. 2658–2713, 2018, doi: 10.1109/COMST.2018.2843533.

[74] B. Saltaformaggio *et al.*, "Eavesdropping on fine-grained user activities within smartphone apps over encrypted network traffic," *10th USENIX Work. Offensive Technol. WOOT 2016*, 2016.

[75] E. Vanrykel, G. Acar, M. Herrmann, and C. Diaz, "Leaky birds: Exploiting mobile application traffic for surveillance," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 9603 LNCS, pp. 367–384, 2017, doi: 10.1007/978-3-662-54970-4_22.

[76] R. Spreitzer, S. Griesmayr, T. Korak, and S. Mangard, "Exploiting data-usage statistics for website fingerprinting attacks on android," *WiSec 2016 - Proc. 9th ACM Conf. Secur. Priv. Wirel. Mob. Networks*, pp. 49–60, 2016, doi: 10.1145/2939918.2939922.

[77] Nancy, "Android Malware Detection using Decision Trees and Network Traffic," *Int. J. Comput. Sci. Inf. Technol.*, vol. 7, no. 4, pp. 1970–1974, 2016, [Online]. Available:

http://ijcsit.com/docs/Volume 7/vol7issue4/ijcsit2016070467.pdf.

[78] E. Papadogiannaki and S. Ioannidis, “A Survey on Encrypted Network Traffic Analysis Applications, Techniques, and Countermeasures,” *ACM Comput. Surv.*, vol. 54, no. 6, 2021, doi: 10.1145/3457904.

[79] I. Voronov and K. Gnezdilov, “Determining OS and Applications by DNS Traffic Analysis,” *Proc. 2021 IEEE Conf. Russ. Young Res. Electr. Electron. Eng. ElConRus 2021*, pp. 72–76, 2021, doi: 10.1109/ElConRus51938.2021.9396085.

[80] S. Mongkolluksamee, V. Visoottiviseth, and K. Fukuda, “Combining communication patterns & traffic patterns to enhance mobile traffic identification performance,” *J. Inf. Process.*, vol. 24, no. 2, pp. 247–254, 2016, doi: 10.2197/ipsjjip.24.247.

[81] V. F. Taylor, R. Spolaor, M. Conti, and I. Martinovic, “Robust Smartphone App Identification via Encrypted Network Traffic Analysis,” *IEEE Trans. Inf. Forensics Secur.*, vol. 13, no. 1, pp. 63–78, 2018, doi: 10.1109/TIFS.2017.2737970.

[82] Y. Huang, B. Li, M. Barni, and J. Huang, “Identification of VoIP Speech with Multiple Domain Deep Features,” *IEEE Trans. Inf. Forensics Secur.*, vol. 15, no. c, pp. 2253–2267, 2020, doi: 10.1109/TIFS.2019.2960635.

[83] G. Aceto, D. Ciuonzo, A. Montieri, and A. Pescapé, “Mobile encrypted traffic classification using deep learning: Experimental evaluation, lessons learned, and challenges,” *IEEE Trans. Netw. Serv. Manag.*, vol. 16, no. 2, pp. 445–458, Jun. 2019, doi: 10.1109/TNSM.2019.2899085.

[84] N. Malik, J. Chandramouli, P. Suresh, K. Fairbanks, L. Watkins, and W. H. Robinson, “Using network traffic to verify mobile device forensic artifacts,” *2017 14th IEEE Annu. Consum. Commun. Netw. Conf. CCNC 2017*, pp. 114–119, 2017, doi: 10.1109/CCNC.2017.7983091.

[85] S. E. Coull and K. P. Dyer, “Traffic Analysis of Encrypted Messaging Services,” *ACM SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 5, pp. 5–11, 2014, doi: 10.1145/2677046.2677048.

[86] Y. Song and U. Hengartner, “PrivacyGuard: A VPN-based platform to detect information leakage on android devices,” *SPSM 2015 - Proc. 5th Annu. ACM CCS Work. Secur. Priv. Smartphones Mob. Devices, co-located with CCS 2015*, pp. 15–26, 2015, doi: 10.1145/2808117.2808120.

[87] F. A. Narudin, A. Feizollah, N. B. Anuar, and A. Gani, “Evaluation of machine learning classifiers for mobile malware detection,” *Soft Comput.*, vol. 20, no. 1, pp. 343–357, 2016, doi: 10.1007/s00500-014-1511-6.

[88] S. Wang *et al.*, “TrafficAV: An effective and explainable detection of mobile malware behavior

using network traffic," *2016 IEEE/ACM 24th Int. Symp. Qual. Serv. IWQoS 2016*, 2016, doi: 10.1109/IWQoS.2016.7590446.

[89] A. Arora and S. K. Peddoju, "NTPDroid: A Hybrid Android Malware Detector Using Network Traffic and System Permissions," *Proc. - 17th IEEE Int. Conf. Trust. Secur. Priv. Comput. Commun. 12th IEEE Int. Conf. Big Data Sci. Eng. Trust. 2018*, pp. 808–813, 2018, doi: 10.1109/TrustCom/BigDataSE.2018.00115.

[90] A. Zulkifli, I. R. A. Hamid, W. M. Shah, and Z. Abdullah, "Android malware detection based on network traffic using decision tree algorithm," *Adv. Intell. Syst. Comput.*, vol. 700, pp. 485–494, 2018, doi: 10.1007/978-3-319-72550-5_46.

[91] Y. Pang *et al.*, "Finding Android Malware Trace from Highly Imbalanced Network Traffic," in *Proceedings - 2017 IEEE International Conference on Computational Science and Engineering and IEEE/IFIP International Conference on Embedded and Ubiquitous Computing, CSE and EUC 2017*, Aug. 2017, vol. 1, pp. 588–595, doi: 10.1109/CSE-EUC.2017.108.

[92] Z. Cheng, X. Chen, Y. Zhang, S. Li, and Y. Sang, "Detecting Information Theft Based on Mobile Network Flows for Android Users," *2017 IEEE Int. Conf. Networking, Archit. Storage, NAS 2017 - Proc.*, 2017, doi: 10.1109/NAS.2017.8026853.

[93] L. Watkins, A. L. Kalathummarath, and W. H. Robinson, "Network-based detection of mobile malware exhibiting obfuscated or silent network behavior," *CCNC 2018 - 2018 15th IEEE Annu. Consum. Commun. Netw. Conf.*, vol. 2018-Janua, pp. 1–4, 2018, doi: 10.1109/CCNC.2018.8319162.

[94] S. Wang *et al.*, "Deep and Broad Learning Based Detection of Android Malware via Network Traffic," *2018 IEEE/ACM 26th Int. Symp. Qual. Serv. IWQoS 2018*, 2019, doi: 10.1109/IWQoS.2018.8624143.

[95] S. Wei, P. Jiang, Q. Yuan, and J. Wang, "Mobile Application Network Behavior Detection and Evaluation with WGAN and Bi-LSTM," *IEEE Reg. 10 Annu. Int. Conf. Proceedings/TENCON*, vol. 2018-Octob, no. 61472189, pp. 44–49, 2019, doi: 10.1109/TENCON.2018.8650372.

[96] A. Arora, S. Garg, and S. K. Peddoju, "Malware detection using network traffic analysis in android based mobile devices," in *Proceedings - 2014 8th International Conference on Next Generation Mobile Applications, Services and Technologies, NGMAST 2014*, Dec. 2014, pp. 66–71, doi: 10.1109/NGMAST.2014.57.

[97] A. Shabtai, L. Tenenboim-Chekina, D. Mimran, L. Rokach, B. Shapira, and Y. Elovici, "Mobile

malware detection through analysis of deviations in application network behavior," *Comput. Secur.*, vol. 43, pp. 1–18, 2014, doi: 10.1016/j.cose.2014.02.009.

[98] A. Arora and S. K. Peddoju, "Minimizing network traffic features for Android mobile malware detection," Jan. 2017, doi: 10.1145/3007748.3007763.

[99] S. Esmaeili and H. R. Shahriari, "PodBot: A New Botnet Detection Method by Host and Network-Based Analysis," *ICEE 2019 - 27th Iran. Conf. Electr. Eng.*, pp. 1900–1904, 2019, doi: 10.1109/IranianCEE.2019.8786432.

[100] A. Liu, Z. Chen, S. Wang, L. Peng, C. Zhao, and Y. Shi, "A fast and effective detection of mobile malware behavior using network traffic," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2018, vol. 11337 LNCS, pp. 109–120, doi: 10.1007/978-3-030-05063-4_10.

[101] L. Liu, X. Huang, A. Zhou, P. Jia, and L. Liu, "Fuzzing the Android Applications with HTTP/HTTPS Network Data," *IEEE Access*, vol. 7, pp. 59951–59962, 2019, doi: 10.1109/ACCESS.2019.2915339.

[102] S. Wang *et al.*, "Deep and broad URL feature mining for android malware detection," *Inf. Sci. (Ny).*, vol. 513, pp. 600–613, Mar. 2020, doi: 10.1016/j.ins.2019.11.008.

[103] A. Arora and S. K. Peddoju, "NTPDroid: A Hybrid Android Malware Detector Using Network Traffic and System Permissions," in *Proceedings - 17th IEEE International Conference on Trust, Security and Privacy in Computing and Communications and 12th IEEE International Conference on Big Data Science and Engineering, Trustcom/BigDataSE 2018*, Sep. 2018, pp. 808–813, doi: 10.1109/TrustCom/BigDataSE.2018.00115.

[104] Z. Li, L. Sun, Q. Yan, and W. Srisa-an, "DroidClassi fi er : Ef fi cient Adaptive Mining of Application-Layer Header for Classifying Android Malware," vol. 1, pp. 597–616, 2017, doi: 10.1007/978-3-319-59608-2.

[105] Q. Li *et al.*, *MulAV: Multilevel and explainable detection of android malware with data fusion*, vol. 11337 LNCS, no. 2016. Springer International Publishing, 2018.

[106] X. Su, J. Lin, F. Shen, and Y. Zheng, *Two-phases detection scheme: Detecting android malware in android markets*, vol. 842. Springer International Publishing, 2019.

[107] A. Zulkifli, I. R. A. Hamid, W. M. Shah, and Z. Abdullah, "Android malware detection based on network traffic using decision tree algorithm," in *Advances in Intelligent Systems and Computing*, 2018, vol. 700, pp. 485–494, doi: 10.1007/978-3-319-72550-5_46.

[108] J. Malik and R. Kaushal, “CREDROID: Android malware detection by network traffic analysis,” in *PAMCO 2016 - Proceedings of the 2nd MobiHoc International Workshop on Privacy-Aware Mobile Computing*, Jul. 2016, pp. 28–36, doi: 10.1145/2940343.2940348.

[109] S. Wang *et al.*, “Deep and broad URL feature mining for android malware detection,” *Inf. Sci. (Ny).*, vol. 513, pp. 600–613, 2020, doi: 10.1016/j.ins.2019.11.008.

[110] S. Wang, Z. Chen, Q. Yan, B. Yang, L. Peng, and Z. Jia, “A mobile malware detection method using behavior features in network traffic,” *J. Netw. Comput. Appl.*, vol. 133, pp. 15–25, May 2019, doi: 10.1016/j.jnca.2018.12.014.

[111] I. J. Sanz and M. A. Lopez, “A Lightweight Network-based Android Malware Detection System,” pp. 695–703, 2020.

[112] M. Alshehri, “APP - NTS : a network traffic similarity - based framework for repacked Android apps detection,” no. Aleieldin 2018, pp. 1–10, 2021.

[113] V. Sihag, G. Choudhary, M. Vardhan, P. Singh, and J. T. Seo, “PICAndro : Packet InspeCtion-Based Android Malware Detection,” vol. 2021, 2021.

[114] A. Arora and S. K. Peddoju, “NTPDroid: A Hybrid Android Malware Detector Using Network Traffic and System Permissions,” *Proc. - 17th IEEE Int. Conf. Trust. Secur. Priv. Comput. Commun. 12th IEEE Int. Conf. Big Data Sci. Eng. Trust. 2018*, pp. 808–813, 2018, doi: 10.1109/TrustCom/BigDataSE.2018.00115.

[115] “Baidu Mobile Assistant 2020.” [Online]. Available: https://shouji.baidu.com/.

[116] S. Kandukuru and R. M. Sharma, *Android malicious application detection using permission vector and network traffic analysis*, vol. 2017-Janua. 2017.

[117] S. Nepal and M. V Ramakrishna, “NeSeDroid- Android Malware Detection Based on Network Traffic and Sensitive Resource Accessing,” in *Proceedings of the International Conference on Data Engineering*, 1999, pp. 22–31, doi: 10.1007/978-981-10-1678-3.

[118] A. H. Lashkari, A. F. Akadir, H. Gonzalez, K. F. Mbah, and A. A. Ghorbani, “Towards a network-based framework for android malware detection and characterization,” *Proc. - 2017 15th Annu. Conf. Privacy, Secur. Trust. PST 2017*, no. Cic, pp. 233–242, 2018, doi: 10.1109/PST.2017.00035.

[119] Z. Chen *et al.*, “A first look at android malware traffic in first few minutes,” *Proc. - 14th IEEE Int. Conf. Trust. Secur. Priv. Comput. Commun. Trust. 2015*, vol. 1, pp. 206–213, 2015, doi: 10.1109/Trustcom.2015.376.

[120] A. Feizollah, N. B. Anuar, R. Salleh, and F. Amalina, *Comparative study of k-means and mini batch k-means clustering algorithms in android malware detection using network traffic analysis*. 2015.

[121] C. H. Kim, "Security analysis of YKHL distance bounding protocol with adjustable false acceptance rate," *IEEE Commun. Lett.*, vol. 15, no. 10, pp. 1078–1080, 2011, doi: 10.1109/LCOMM.2011.080811.111299.

[122] S. Sumriddetchkajorn and Y. Intaravanne, "Data-nonintrusive photonics-based credit card verifier with a low false rejection rate," *Appl. Opt.*, vol. 49, no. 5, pp. 764–771, 2010, doi: 10.1364/AO.49.000764.

[123] M. E. Schuckers, *Receiver Operating Characteristic Curve and Equal Error Rate*. 2010.

[124] G. O. Williams, "Use of d' as a `decidability' index," *IEEE Annu. Int. Carnahan Conf. Secur. Technol. Proc.*, pp. 65–69, 1996.

[125] W. Zhu, N. Zeng, and N. Wang, "Sensitivity, specificity, accuracy, associated confidence interval and ROC analysis with practical SAS® implementations.," *Northeast SAS Users Gr. 2010 Heal. Care Life Sci.*, pp. 1–9, 2010.

[126] S. Y. Yerima, S. Sezer, and I. Muttik, "High accuracy android malware detection using ensemble learning," *IET Inf. Secur.*, vol. 9, no. 6, pp. 313–320, 2015, doi: 10.1049/iet-ifs.2014.0099.

[127] H. Huang, H. Xu, X. Wang, and W. Silamu, "Maximum F1-score discriminative training criterion for automatic mispronunciation detection," *IEEE/ACM Trans. Audio Speech Lang. Process.*, vol. 23, no. 4, pp. 787–797, 2015, doi: 10.1109/TASLP.2015.2409733.

[128] D. Arp, M. Spreitzenbarth, M. Hübner, H. Gascon, and K. Rieck, "Drebin: Effective and Explainable Detection of Android Malware in Your Pocket," no. February, pp. 23–26, 2014, doi: 10.14722/ndss.2014.23247.

[129] F. Wei, Y. Li, S. Roy, X. Ou, and W. Zhou, "Deep ground truth analysis of current android malware," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 10327 LNCS, pp. 252–276, 2017, doi: 10.1007/978-3-319-60876-1_12.

[130] "GOOGLE PLAY STORE, 2021." [Online]. Available: https://play.google.com/store.

[131] L. Li *et al.*, "AndroZoo++: Collecting millions of android apps and their metadata for the research community," *arXiv*, pp. 468–471, 2017.

[132] S. Mahdavifar, A. F. Abdul Kadir, R. Fatemi, D. Alhadidi, and A. A. Ghorbani, "Dynamic Android Malware Category Classification using Semi-Supervised Deep Learning," *Comput. IEEE 6th Int.*

Conf. Cloud Big Data Comput. IEEE 5th Cybe, pp. 515–522, 2020, doi: 10.1109/DASC-PICom-CBDCom-CyberSciTech49142.2020.00094.

[133] S. Kumar, S. Indu, and G. S. Walia, "Recent Advances in Android Malware Detection- A Survey.," 2020. *Proc. - First Int. Conf. on Communication , Computing and Signal Processing*, NIT, Jalandhar, Punjab,India

[134] S. Kumar, S. Indu, and G. S. Walia, "Smartphone Traffic Analysis: A Contemporary Survey of the State-of-the-Art," in *Advances in Intelligent Systems and Computing*, 2021, vol. 1262, pp. 325–343, doi: 10.1007/978-981-15-8061-1_26.

[135] Hardware Features, https://developer.android.com/ guide/topics/manifest/uses-feature-element#hw-features (visited on May 20,2022)

[136] López, Christian Camilo et. al. "Features to Detect Android Malware." *2018 IEEE Colombian Conference on Communications and Computing (COLCOM)* (2018): 1-6.

[137] G. S. Walia, H. Ahuja, A. Kumar, N. Bansal, and K. Sharma, "Unified Graph-Based Multicue Feature Fusion for Robust Visual Tracking," *IEEE Trans. Cybern.*, vol. 50, no. 6, pp. 2357–2368, 2020, doi: 10.1109/TCYB.2019.2920289.

[138] T. Denœux and M.-H. Masson, "Dempster-Shafer Reasoning in Large Partially Ordered Sets: Applications in Machine Learning," pp. 39–54, 2010, doi: 10.1007/978-3-642-11960-6_5.

[139] B. Wang, J. Jiang, W. Wang, Z. H. Zhou, and Z. Tu, "Unsupervised metric fusion by cross diffusion," *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, pp. 2997–3004, 2012, doi: 10.1109/CVPR.2012.6248029.

[140] T. Tong, K. Gray, Q. Gao, L. Chen, and D. Rueckert, "Multi-modal classification of Alzheimer's disease using nonlinear graph fusion," *Pattern Recognit.*, vol. 63, pp. 171–181, 2017, doi: 10.1016/j.patcog.2016.10.009.

[141] Y. Du, X. Wang, and J. Wang, "A static Android malicious code detection method based on multi-source fusion," *Secur. Commun. Networks*, vol. 8, no. 17, pp. 3238–3246, Nov. 2015, doi: 10.1002/sec.1248.

[142] F. Xiao, "A new divergence measure for belief functions in D–S evidence theory for multisensor data fusion," *Inf. Sci. (Ny).*, vol. 514, no. xxxx, pp. 462–483, 2020, doi: 10.1016/j.ins.2019.11.022.

[143] X. Jiang, B. Mao, J. Guan, and X. Huang, "Android Malware Detection Using Fine-Grained Features," *Sci. Program.*, vol. 2020, 2020, doi: 10.1155/2020/5190138.

[144] X. Wang, D. Zhang, X. Su, and W. Li, “Mlifdect: Android malware detection based on parallel machine learning and information fusion,” *Secur. Commun. Networks*, vol. 2017, 2017, doi: 10.1155/2017/6451260.

[145] S. Kumar, S. Indu, and G. S. Walia, “An Efficient Multistage Fusion Approach for Smartphone Security Analysis,” *Def. Sci. J.*, vol. 71, no. 4, pp. 476–490, 2021, doi: 10.14429/dsj.71.15077.

[146] “Statista, Application in Google Playstore.,” 2020. https://www.statista.com/statistics/266210/number-of-available-applications-in-the-google-play-store/ (accessed Feb. 20, 2021).

[147] “Statista, Development of new Android malware worldwide from June 2016 to March 2020,” 2020. https://www.statista.com/statistics/680705/global-android-malware-volume/ (accessed Feb. 20, 2021).

[148] “CrowdStrike Mobile Threat Report.,” 2020. https://www.crowdstrike.com/blog/mobile-threat-report-2019-trends-and-recommendations/ (accessed Feb. 20, 2020).

[149] Y. Bai and D. Wang, “Fundamentals of Fuzzy Logic Control — Fuzzy Sets, Fuzzy Rules and Defuzzifications BT - Advanced Fuzzy Logic Technologies in Industrial Applications,” pp. 17–36, 2006.

[150] M. Mizumoto, “Defuzzification methods,” *Handb. Fuzzy Comput.*, pp. 1–7, 2004, doi: 10.1887/0750304278/b438c24.

[151] S. Kumar, S. Indu, and G. S. Walia, “Optimal Unification of Static and Dynamic Features for Smartphone Security Analysis,” *Intell. Autom. Soft Comput.*, vol. 33, no. 4, 2022, doi: 10.32604/iasc.2022.024469.

[152] “Percentage of Mobile Device Traffic, 2021, Statistica.” https://www.statista.com/statistics/277125/share-of-website-traffic-coming-from-mobile-devices/ (accessed Jul. 06, 2021).

[153] J. Smarandache, Florentin & Dezert, *Advances and Applications of DSmT for Information Fusion, American Research Press, 4*. 2015.

[154] A. Liu, Z. Chen, S. Wang, L. Peng, C. Zhao, and Y. Shi, *A fast and effective detection of mobile malware behavior using network traffic*, vol. 11337 LNCS, no. 2016. Springer International Publishing, 2018.

[155] S. Kumar, S. Indu, and G. G. . Walia, “A Novel Traffic based Approach for Smartphone Security Analysis,” *Def. Sci. J.*, vol. 72, no. 2, 2022, doi: 10.14429/dsj.72.17522.

Appendix-A

List of Publications

1. **Sumit Kumar**, S. Indu, Gurjit Singh Walia (2021), "**An Efficient Multistage Fusion Approach for Smartphone Security analysis**", Defence Science Journal, **SCIE**, Vol. 71(4), pp 476-490, DOI:10.14429/dsj.71.15077

2. **Sumit Kumar**, S. Indu, Gurjit Singh Walia (2022), "**Optimal Unification of Static and Dynamic Features for Smartphone Security Analysis**", Journal of Intelligent Automation and Soft Computing, **SCIE**, Vol. 35(1), pp 1035-1051 DOI: 10.32604/iasc.2022.024469.

3. **Sumit Kumar**, S. Indu, Gurjit Singh Walia (2022)**, "A Novel Traffic based Framework for Smartphone Security Analysis"** Defence Science Journal, **SCIE ,** vol. 72(3), pp 371-382, DOI: 10.14429/dsj.72.17522.

4. **Sumit Kumar**, S. Indu, Gurjit Singh Walia (2020), "**Smartphone Traffic Analysis: A Contemporary Survey of the State-of-the-Art**", International Conference on Mathematics and Computing - (ICMC-2020) DOI: 10.1007/978-981-15-8061-1_26 44

5. **Sumit Kumar**, S. Indu, Gurjit Singh Walia (2020), "**Recent Advances in Android Malware Detection- A Survey**", International Conference on Communication, Computing and Signal Processing - (CCSP 2020) (Presented and Accepted for Publication.)

Appendix-B

Biodata

Sumit Kumar completed his B.E (First class with distinction) and M.E (First class with distinction) degrees in Electronics and Communication Engineering from **Delhi College of Engineering**, **Delhi University, Delhi** in 1996 and 2004 respectively. Presently, he holds the position of Scientist 'F' in Scientific Analysis Group, DRDO, Ministry of Defence, Government of India. He has worked in the field of Mobile Security, Secure Communication, Machine Learning and Hardware Analysis. He has prepared more than 50 technical reports. He joined Delhi Technological University, New Delhi as a part time Ph.D. Scholar in Electronics and Communication department under the supervision of **Prof. S.Indu and Dr. Gurjit Singh Walia** in 2017. His current research focuses on smartphone security analysis. He has proposed various robust and efficient methods in this research area.