ANDROID MALWARE AND VARIANT DETECTION FRAMEWORK LEVERAGING SIMILARITY HASHES

ALINA KHALID
01-247212-002
PROF. DR. FAISAL BASHIR

A thesis submitted in fulfilment of the requirements for the award of degree of Masters of Science (Information Security)

Department of Computer Science

BAHRIA UNIVERSITY ISLAMABAD

SEPTEMBER 2023

Approval of Examination

Scholar Name: **Alina Khalid**

Registration Number: **75922**

Enrollment: **01-247212-002**

Program of Study: **MS (Information Security)**

Thesis Title: **Android Malware and Variant Detection Framework Leveraging Similarity Hashes**

It is to certify that the above scholar's thesis has been completed to my satisfaction and, to my belief, its standard is appropriate for submission for examination. I have also conducted plagiarism test of this thesis using HEC prescribed software and found similarity index 13 %. that is within the permissible limit set by the HEC for the MS/M.Phil degree thesis. I have also found the thesis in a format recognized by the BU for the MS/M.Phil thesis.

Supervisor Name : Dr.Faisal Bashir

[illegible]

Principal Supervisor Signature

Date: 25th October, 2023

Author's Declaration

I, Alina Khalid hereby state that my MS/M.Phil thesis titled is my own work and has not been submitted previously by me for taking any degree from Bahria university or anywhere else in the country/world. At any time if my statement is found to be incorrect even after my graduation, the University has the right to withdraw/cancel my MS/M.Phil degree.

Name of Scholar: **Alina Khalid**

Date: **17th September, 2023**

Plagiarism Undertaking

I, solemnly declare that research work presented in the thesis titled **Android Malware & Variant Detection Framework Leveraging Similarity Hashes** is solely my research work with no significant contribution from any other person. Small contribution / help wherever taken has been duly acknowledged and that complete thesis has been written by me. I understand the zero tolerance policy of the HEC and Bahria University towards plagiarism. Therefore I as an author of the above titled thesis declare that no portion of my thesis has been plagiarized and any material used as reference is properly referred / cited.

I undertake that if I am found guilty of any formal plagiarism in the above titled thesis even after award of MS/M.Phil degree, the university reserves the right to withdraw / revoke my MS/M.Phil degree and that HEC and the University has the right to publish my name on the HEC / University website on which names of scholars are placed who submitted plagiarized thesis.

Name of Scholar: **Alina Khalid**

Date: **17th September, 2023**

Dedication

With heartfelt gratitude, I dedicate this thesis to my beloved parents, whose unwavering support, encouragement, and love have been my guiding light throughout this academic journey. To my siblings and extended family, whose words of wisdom and unwavering belief in me have been a constant source of inspiration. I extend my deepest appreciation to my dedicated mentor, friends, and classmates for their valuable insights and camaraderie. Above all, I express my deepest thanks to Allah Almighty, the Most Merciful and Most Compassionate for His guidance, blessings, and unwavering support that have been my ultimate source of strength. I am humbled and grateful for the opportunities and abilities Allah Almighty has bestowed upon me.

Alina Khalid

Acknowledgements

In preparing this thesis, I was in contact with many people, researchers, academicians, and practitioners. They have contributed towards my understanding and thoughts. In particular, I wish to express my sincere appreciation to my main thesis supervisor, **Professor Dr. Faisal Bashir**, for encouragement, motivation and guidance. Without his continued support and interest, this thesis would not have been the same as presented here.

Librarians at Bahria University also deserve special thanks for their assistance in supplying the relevant literatures. My fellow postgraduate students should also be recognised for their support. My sincere appreciation also extends to all my colleagues and others who have provided assistance at various occasions. Their views and tips are useful indeed. Unfortunately, it is not possible to list all of them in this limited space. I am grateful to all my family members.

Abstract

In the growing world of technology, the frequency and magnitude of cyber attacks is increasing day by day. Most popular OS are most prone to these attacks. Android OS is sharing a major share in OS market therefore facing the challenge of frequent and sophisticated malware attacks. These malware are created in a way to bypass network security systems. Major categories of malware remain the same however small modifications in malware can make it act differently and hence challenging to identify. Various techniques and algorithms are used for the identification and categorization of these variants to make better security and incidence responses. Fuzzy hashes are used to calculate the similarity index between files to identify malicious sections inside an appearing legitimate file. In this paper, research has been conducted to evaluate and improve the working, accuracy, and reliability of fuzzy hashes of static features of APK files in detecting Android malware and classifying its variants. In contrast to conventional research methodologies, our study adopts a distinctive static feature-based fuzzy hashing technique for the detection of malware and its variants. This approach has enabled us to achieve promising results in our experiments. We selected a dataset consisting of 2000 APK files, containing both malicious and benign samples. For variant identification and family classification, we've selected random malware families from six distinct categories: trojan, adware, spyware, virus, downloader, and hacktool. Through rigorous experimentation, our findings have demonstrated a significant improvement in key metrics such as precision, recall, and the F-Measure. These improvements collectively contribute to an overall enhancement in the accuracy, reaching 96.67%, all without the dependence on intricate machine learning or deep learning methods.

TABLE OF CONTENTS

LIST OF TABLE

LIST OF FIGURE

CHAPTER 1

INTRODUCTION

In the realm of cybersecurity, Android malware has become a prevalent and evolving threat. In this chapter, we delve into the multifaceted landscape of Android malware and its variants, employing fuzzy or similarity hashes as our primary tool for detection. We begin by exploring the various types of Android malware, emphasizing the need for robust detection methods. We also dissect the key components of an APK file, the fundamental unit of Android apps. With a strong foundation in malware analysis and detection techniques, we present the problem background and articulate the problem statement that guides our research.

In market distributions of smartphone OS, the most dominant one is the Android operating system. According to a report generated by StatCounter GlobalStats, the market share ratio of Android OS is leading by 70.77% [2]. The increase in popularity and dependency on the Android platform is demanding an increase in security from malicious scripts and applications. However, with the advancement of technologies malware are also becoming sophisticated and are difficult to detect. A report published by Statista Research Department in July 2022 shows that from March 2020, new Android malware samples reached the number of 482,579 per month. According to AV-Test, Trojans were the most common type of malware affecting Android devices [3]

Malware (malicious software) are modified to generate its variants. Though the major section of the variant is similar to its parent malware however it is difficult to identify its malware family from behavioral analysis of the variant. Researchers are working on various dimensions to detect and identify Android malware and variants. The main research areas that are being widely used contain malware analysis using dynamic, static, and hybrid approaches [4]. Existing work on dynamic Android malware analysis shows that it provides reliable and accurate results against obfuscated Android malware and variants, however, dynamic analysis requires an emulator to monitor the run-time activities of an APK file to identify the abnormal and malicious behavior of the file. However complex malware are programmed in a way that whenever they detect these dummy environments they behave as normal or benign software. Moreover, Dynamic analysis requires high consumption of resources and latency rate [5]. On the other hand static analysis provides accuracy with limited resources and optimum latency, as it extracts features such as permissions, API calls, opcode sequences, and grey/colored images from APK files to perform further analysis without running it on any emulator [5]. Therefore static analysis requires fewer resources to analyze malware and goodware. In Hybrid analysis, both static and dynamic analysis are combined to obtain better accuracy and optimal results, however hybrid analysis also requires high computational cost to incorporate dynamic features analysis. Research have been conducted on various techniques including feature extraction via opcode sequences, analyzing call graphs, performing comparison on files using cryptography to detect malware variant. Our scope of research is to identify malware variants within Android package files and to classify these variants into specific malware families by using the approach of static feature-based hashing on APK files. To achieve this, we utilize cryptographic hashes for identification and similarity hashes for classification of variant into malware family.

In recent works[6][7][8], identification and classification of malware and its variants is performed using cryptographic techniques like fuzzy hash and cryptographic hash that are applied on file and its sections to compare the level of mutation and similarity between PE files. A similar technique can be used to identify Android malware and variants by comparing sections of APK files with files present in Android malware database. The similarity ratio of both files can be calculated to identify whether a given APK file is a malware variant or benign software. Hence lightweight mechanism for malware detection and variant classification can be achieved.

1.1 Types of Android Malware

Malware in the Android ecosystem encompasses a variety of malicious software designed to compromise user data, device integrity, and privacy. The following sections outline common categories of Android malware:

1.1.1 Adware

Adware, short for advertising-supported software, capitalizes on the delivery of intrusive advertisements to generate revenue for its creators. These advertisements disrupt user experiences and often collect sensitive user data for targeted ad delivery.

1.1.2 Riskware

Riskware refers to applications that may not exhibit outright malicious behavior but carry potential risks due to vulnerabilities or intrusive data collection practices. Such applications can inadvertently expose users to security threats.

1.1.3 Spyware

Spyware covertly monitors and records user activities, including keystrokes, browsing history, and multimedia interactions. This category poses severe threats to user privacy, as personal and sensitive information is harvested without consent.

1.1.4 Banking Malware

Banking malware specifically targets users' financial data and credentials. By intercepting sensitive information, these malicious applications facilitate unauthorized access to bank accounts, leading to financial loss and identity theft.

1.1.5 SMS Malware

SMS malware, also known as SMS Trojans, manipulate devices to send unauthorized premium SMS messages to premium-rate numbers. This results in unexpected charges for users, financially benefiting attackers.

1.2 Components of Android Package File

An Android Package (APK) file is the fundamental unit of installation for Android applications. It comprises various components that collectively define an app's behavior and appearance. Figure 1.1 provides an overview of the composition of an APK file, offering insight into the primary components and the main content and features encapsulated within each of them. Brief detail of these components is given below:

1.2.1 Manifest.xml

The AndroidManifest.xml file is a critical component of the APK, providing vital metadata about the application. It outlines the app's components, permissions required, and hardware features necessary for proper functionality.

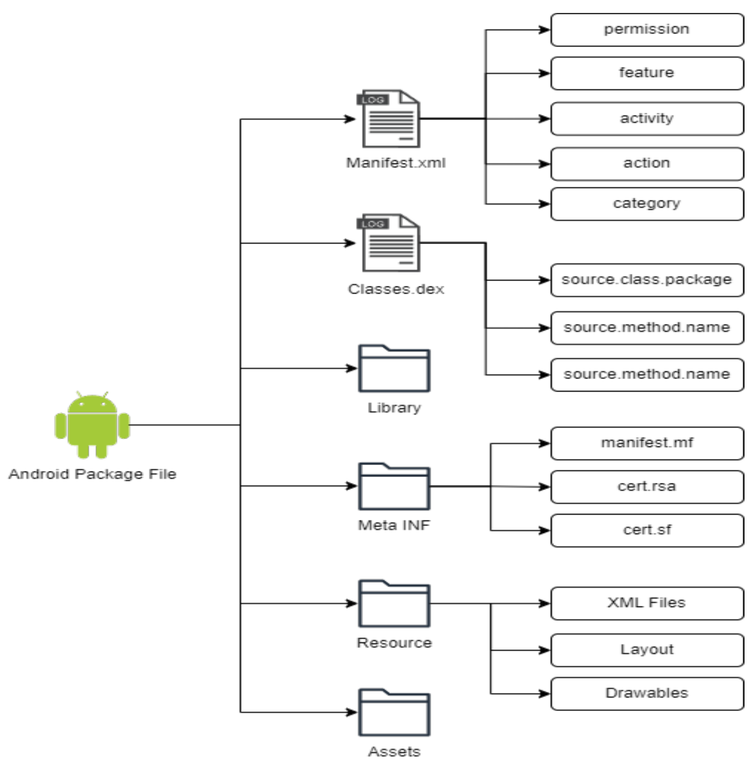

Figure 1.1: Structure of an APK file with its major components

The manifest file plays a crucial role in the Android system's management of the application.

1.2.2 Classes.dex

The classes.dex file contains the compiled bytecode of the application's Java source code. This bytecode is executed by the Dalvik Virtual Machine (pre-Android 5.0) or the Android Runtime (ART, Android 5.0 and later). The efficient execution of bytecode enables app functionality and logic.

1.2.3 Resource Folder

The res folder holds essential resources that contribute to the application's user interface and functionality. These resources include XML layout files, images, strings, and other assets that collectively define the app's appearance and user interaction.

1.2.4 Library Folder

The lib folder contains external libraries and dependencies utilized by the application. These libraries enhance the app's capabilities and provide access to various features without having to reinvent the wheel. Integrating external libraries streamlines development and ensures code quality.

1.2.5 Meta Inf

The META-INF folder contains metadata essential for verifying the integrity and authenticity of the APK. This metadata includes digital signatures that assure users the app has not been tampered with and originates from a trusted source.

1.3 Malware Analysis

To combat the growing threat of Android malware, researchers and cybersecurity experts employ various analysis techniques to dissect and understand the behavior of malicious applications. These analysis methods provide insights into the inner workings of malware, aiding in the development of effective detection and prevention strategies.Figure 1.2 provides an in-depth exploration of malware analysis techniques, highlighting the prevalent approaches commonly employed within these methods. These primary approaches to malware analysis are:

1.3.1 Static Analysis

Static analysis involves examining an application's code, resources, and metadata without executing the app. This approach helps identify potential vulnerabilities, malicious behaviors, and security risks. By scrutinizing the

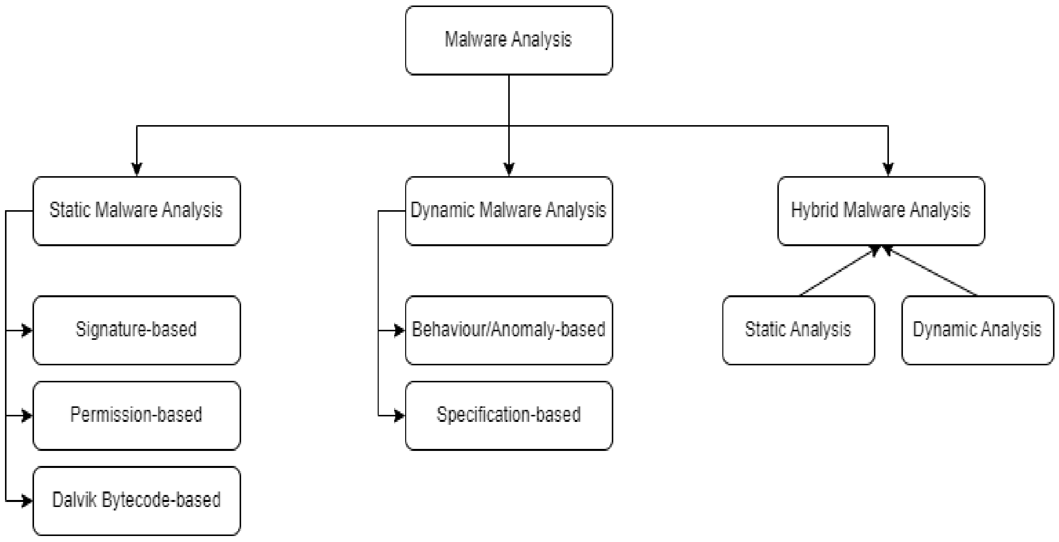

Figure 1.2: Taxonomy of Malware Analysis

app's source code, permissions, and data flows, static analysis can highlight instances of potential malware.

1.3.2 Dynamic Analysis

Dynamic analysis entails executing the application in a controlled environment to observe its behavior during runtime. This approach captures runtime actions, such as network communication, file access, and system interactions. Dynamic analysis provides insights into the actual behavior of the app, including interactions with external servers and potential exploitation of device resources.

1.3.3 Hybrid Analysis

Hybrid analysis combines aspects of both static and dynamic analysis to provide a comprehensive view of an application's behavior. By leveraging the strengths of each approach, the hybrid analysis aims to overcome limitations and gather a holistic understanding of the malware's capabilities.

1.4 Detection Techniques

As the threat landscape evolves, innovative techniques for detecting Android malware emerge. Following are the prominent detection methodologies:

Image Processing for Malware Detection: Image processing techniques, typically used in computer vision, are employed for detecting patterns and anomalies within an application's visual elements. By analyzing icons, images, and user interface components, researchers aim to identify potential visual indicators of malware presence.

Opcode Sequences and API Call Based Malware Detection: This approach delves into the executable aspect of applications by analyzing opcode sequences and API calls. By studying the low-level machine instructions and the high-level interface interactions, researchers aim to identify patterns indicative of malicious behaviors. This method effectively bridges the gap between static and dynamic analysis.

Malware Detection Using Cryptography: Cryptography-based methods leverage the principles of secure communication and data integrity for malware detection. This section explores two sub-methods within this category:

1.4.1 Cryptographic Hashes

Cryptographic hashes, such as MD5, SHA-1, and SHA-256, create unique digital fingerprints of files. These hashes are used to verify the integrity of files and identify any changes or tampering. Hashes of known good files are compared with those of suspicious files to detect alterations.

1.4.2 Similarity (Fuzzy) Hashes

Similarity hashes, often referred to as fuzzy hashes, calculate a similarity index between files. This approach is particularly valuable for identifying subtle variations in files that might indicate malicious alterations. Notable similarity hash algorithms include:

(a) Ssdeep: It calculates a fuzzy hash based on contextual data, such as byte sequences and block sizes. Ssdeep provides a measure of similarity that can highlight malicious code hidden within legitimate files.

(b) Sdhash: This method computes hash values for sequences of data blocks, allowing it to identify similar content even in the presence of slight modifications.

(c) MvHASH-B: A variant of the Multivariable Hash (MvHASH) algorithm, MvHASH-B creates hashes using multiple features of files, aiding in the identification of common patterns among potentially malicious files.

(d) TLSH: TLSH is a locality-sensitive hash that's particularly effective in identifying similarities in binary data. It's used for comparing binary files and is commonly employed in malware analysis. TLSH generates hash values that represent the structural characteristics of binary data, making it useful for detecting similar code patterns within malware samples.

1.5 Problem Background

Since the popularity of the Android platform is increasing with the passage of time, it is therefore becoming one of the major targets for threat actors. To bypass security mechanisms, malware is modified by attackers to get its variant. Though, the working behavior of malware gets completely changed, yet a major portion of the file remains unchanged. To identify and analyze the properties of malware and variants, dynamic analysis has been widely used. The dynamic analysis provides high accuracy in malware detection and classification and provides less rate of false positive detection. However, a limitation of this approach is that it requires an emulatory dummy environment to analyze the working behavior of malware. This procedure is time-consuming and it requires a high cost of implementation. Moreover, advanced malware are designed to act like benign software whenever they detect a virtual environment and, therefore can easily bypass the system. This makes the dynamic analysis less reliable.

In static analysis, the most popular approach to detect malware is to extract features from a targeted file without actually running it. Then these features are fed as an input to any machine learning model to construct classifiers that provide accuracy in results. However, results can be fabricated or modified using adversarial machine learning, creating a backdoor in the training set, or other techniques like training set poisoning, model theft, etc.

Another major technique that is being used to detect and classify malware and variants detection is the use of cryptography or hashing techniques. Import hashing is used to find the hash of imported libraries, functions, and API which is further useful for detecting malware variants that share similar libraries or functions. Cryptographic hashes are efficient most to performing binary detection. However, it fails on variant detection because of the cryptographic diffusion property. Fuzzy hashes are used to identify similarities between the files. It provides a lightweight and efficient solution to classify variants using targeted files and malicious software.

In the comparison of malware and its variant, though the working behavior of both files may not have any similarity, yet major portion of both files remain similar. Fuzzy hashing algorithms can be used to identify these similarities. Fuzzy hashes are compression functions that use various techniques to identify similarities between files. It uses algorithms to hash a file in sections. Afterward, it compares the similarity of sections and calculates the similarity ratio. Moreover, by further analysis, both changed and unchanged sections of the APK file can be identified. This file-level and section-level analysis and can be used to identify the class of malware.

1.6 Problem Statement

Android Operating Systems are being used widely, they are more vulnerable to being attacked. In addition to complex malware, threat actors also mutate existing malware to form its variants. To ensure the security of the platform, an efficient and lightweight framework is required, that can perform detection and classification of malicious APK files. Fuzzy hashes can detect effectively satisfying the above criteria yet accuracy needs to be improved to detect and classify sophisticated malware and variants.

1.7 Research Objectives

The objectives of the Research include the following.

(a) Construct a malware detection framework capable of binary classification, family classification, and variant detection within Android Package files.

(b) Employ similarity digests and algorithms like ssdeep to dissect and assess specific sections of APK files for similarity with known malware.

(c) Determine the effectiveness of specific APK features and their impact on Similarity Hash-Based Malware Detection.

(d) To explore Noise-Inducing Features in Result Analysis and their impacts on false positive.

1.8 Thesis Organization

The organization of this thesis is as follows: Chapter 2 provides an overview of the existing body of work pertaining to approaches utilized in identifying malware and variants. Chapter 3 offers an in-depth exploration of the methodology, encompassing dataset and malware family details, environment creation, and details on extracted features. Subsequently, Chapter 4 highlights the analysis and findings of the research, while Chapter 5 presents the conclusion and highlights the areas for future research.

CHAPTER 2

RELATED WORK

Our journey through the literature reveals various approaches to malware detection. In this chapter we explore methods that leverage API calls and opcode sequences, shedding light on how these techniques have been instrumental in uncovering malicious behavior. Additionally, we delve into the techniques of detection through colored and gray-scale image processing, highlighting its potential in identifying malware. Furthermore, we scrutinize cryptographic and similarity hashes, enhancing their efficiency in classifying malware variants. This comprehensive review forms the bedrock upon which our research methodology is built.

Research is progressing day by day to get optimum solutions and tools that will secure the system and network from complex and novel malware. for malware detection using static analysis, various approaches are being used. A major approach is to extract relevant features using API call graphs, Dalvik codes, opcode sequences, and system calls, etc. These features are used to train machine learning models to perform detection. Another approach is to identify malware by processing grey-scaled or colored image files of malware and benign software. Classifiers are used to perform binary and family classification. Furthermore, various hashing techniques and algorithms are used to compare files with those present in a database of known malware.

Our literature review has been organized into three distinct domains:

detection via API feature selection, detection using image processing, and detection utilizing cryptographic and similarity hashes. Figure 2.1 visually represents the research papers cited in our study, categorizing them based on these specific areas.

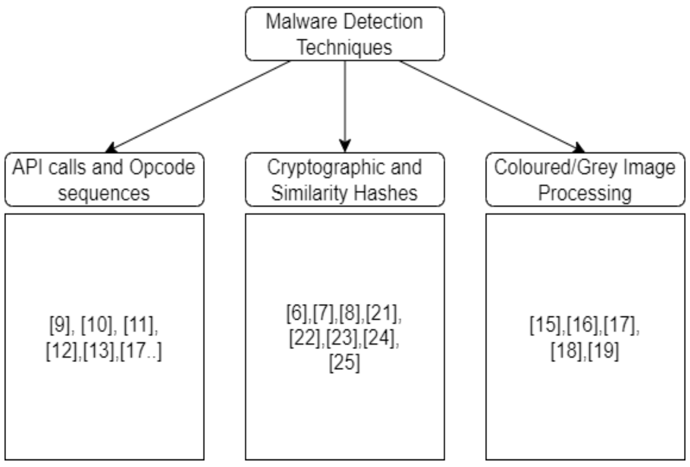
Malware Detection Techniques
API calls and Opcode sequences
Cryptographic and Similarity Hashes
Coloured/Grey Image Processing
[9], [10], [11], [12],[13],[17..]
[6],[7],[8],[21], [22],[23],[24], [25]
[15],[16],[17], [18],[19]

Figure 2.1: Major Malware Detection Approaches

2.0.1 Detection using API Feature Selection

In [9] malware detection is performed by extracting features from APK files and calculating respective weights to identify important features in Android malware. Machine learning models are used to calculate weights and evolution algorithms are used to further optimize the results. A lightweight model was proposed to detect Android malware variants by extracting relevant features from a graph of Dalvik opcodes and pruning non-important edges to minimize complexity. The programs are then compared with pre-labeled malicious files to check whether a program is a malware variant or not. The framework provides an accuracy of 94% and a latency rate of 0.01s for detecting a malicious APK [10]. [11] analyzed all execution paths of instruction call graphs dataset of benign and malicious APK files to improve deep neural learning. Another proposed work detects malware variants by integrating deep learning with image enhancement, global average pooling layer, and Rasnet.

Experimental results showed an accuracy of 96.35% and 94.55% for unobfuscated and obfuscated malware samples respectively. Moreover, accuracy in classification after obfuscation is up to 89.96% [12]. In [13] research work identified detailed characteristics by collecting fine-grained opcode bi-graph and high-level API frequencies from disassembled benign and malicious executables. CNN and BPNN are used to train these opcode and API features. These features are then embedded in the softmax classifier for family classification of malware and variant detection. The model achieved 95% accuracy in malware detection and 90% accuracy in malware family classification. The study in [14] addresses the escalating concerns of malware and piracy in the burgeoning Android app market. It introduces a permission-based malware detection system, which evaluates applications based on their requested permissions. Additionally, the research re-implements Juxtapp, a tool for detecting malware and piracy, focusing on the underlying opcodes of applications. Evaluation involves a substantial dataset, including original, pirated, and malware-infected apps from AndroZoo and KuafuDet. Findings reveal that permission-based malware detection generally outperforms the opcode-based approach. Furthermore, Juxtapp proves effective for detecting software piracy. These combined approaches enhance security and authenticity in the Android app ecosystem. Table 2.1 presented in the literature review highlights research endeavors in the domain of malware detection employing API call graphs. It categorizes these research efforts based on their focus, encompassing binary and family classification as well as variant detection.

Table 2.1: Summary of related works on malware detection and variant classification using API Features and Call Graphs

NO	Technique	Dataset	File Type	Malware Families or Categories	Variant Identification	Classification Type	Accuracy
[10]	Opcode Analysis	GooglePlay Drebin	Android	Fake Installer DroidKungfu Opfake	Yes	Binary & Family	94%
[13]	Opcode Sequences Api Frequency Analysis	VxHeaven	Windows	Backdoor Trojan Droper Trojan Banker Worm	yes	Binary & Family	90%
[11]	Feature Extraction Via Instruction Call Graph	AndroZoo The Argus Group	Android By Argus Group	Undefined	No	Binary & Family	91.42%
[9]	Feature Weighting Function (KNN)	Drebin AMD GooglePlay	Android	Undefined	No	Binary & Family	Improved
[12]	Opcodes Feature Extraction	Drebin	Android	Opfake FakeDoc FakeRun	Yes	Binary & Family	94.55%

2.0.2 Detection using Image Processing

Work proposed in [15] presented a method that converts android malware dex file into colored image and extracts relevant features using CNN to detect and classify android malware and its variants. To perform classification SVM and RF algorithms are used and results showed an accuracy of 100% in binary, and 92.19% in 5-class classification. However, this method is not resilient against adversarial attacks performed on ML and DL. [16] implemented a visual analysis technique to analyze binary executable files after converting them into a 2D grey-scale image. The proposed model, with reduced complexity, gives accurate results on even unknown, zero-day, and obfuscated malware of trained families but can give false predictions on the malware of untrained families. In [17], malware detection and family identification are performed by converting binary files into colored images. Experimental results showed that colored images give more accurate results than grey-scale images. The model is also trained to detect hidden code and obfuscated malware with a lower latency rate. Research in [18] proposed a framework that uses colored label boxes to highlight different sections of malware in PE files. To perform malware classification they created a model that uses image processing, VGG16, and SVM techniques that produce accuracy up to 96.59% and 98.94% on two databases. A semi-supervised method is introduced in [19] for the detection of obfuscated malware. This approach combines deep learning, feature engineering, and image processing techniques. It was compared to traditional gray-scale image-based classification methods, and experiments were conducted to validate its effectiveness. The proposed approach claims to achieve an accuracy rate of 99.12% in detecting obfuscated malware. Another approach to malware detection called Orthogonal Malware Detection (OMD) is introduced in [20]. OMD combines various types of features, including audio descriptors, image similarity descriptors, and static/statistical features, to effectively identify malware. The study demonstrates the effectiveness of audio descriptors in classifying malware families when representing malware binaries as audio signals. It also shows that predictions based on audio descriptors are independent of predictions made using image similarity descriptors and other static features. The paper introduces a framework for assessing the orthogonality of new feature sets compared to existing ones, enabling the integration of new features and detection methods. Experimental results on malware datasets claim the robustness of this approach. Table 2.2 highlights research efforts in the domain of malware detection utilizing image processing providing an overview of studies focused on binary and family classification, as well as variant detection within this field.

Table 2.2: Summary of related works on malware detection and variant classification using Image Processing

NO	Technique	Dataset	File Type	Malware Families	Variant Identification	Classification Type	Accuracy
[17]	2D Image Processing	Maligm Microsoft Malevis	Windows	Agent-fyi Injector Neshta	Yes	Binary	98.65%
[16]	API Calls Opcode N- Gram Strings Control Flow Graphs	Maligm IOT Android Mobile Datase	Android	Agent.FYI Rbot!gen Allaple.L	Yes	Binary & Family	97.85%
[15]	Image Processing	Malnet	Android	Adsware Clicker+ Trojan spyware	Yes	Binary	92.29%
[18]	Colored Image Processing	Vxheaven, 2010 Virusshare, 2010 Microsoft, 2015	PE	-	Yes	Binary & Family	96.59%
[19]	Feature Extraction Via Opcodes sequences	Microsoft Big 2015	exe	-	Yes	Binary & Family	99.12%

2.0.3 Detection using Cryptographic Hashes

Using cryptography in malware detection and variant classification also plays a vital role in giving lightweight and accurate solutions. In [21] a framework is proposed that integrates cryptography, machine learning, natural language, and image processing to give a cloud-based solution for Android malware detection with an accuracy of more than 98.5%. The model proposed in [7] uses similarity digests of opcode sequences taken from Dalvik code to identify similarity in analyzing and malicious APK file. Similarity digests are created from fuzzy hash based on ssdeep. The results are further analyzed by N-gram tokenization. The results help in identifying malware variants, however, their accuracy is compromised when malware are packed with the same packer, therefore improvement is required in this field. Androsimilar [8] is a framework that uses signatures of robust features for malware detection and identifying unknown malware variants. The approach also deals against code obfuscation and repackaging. To improve accuracy results it used syntactic similarity of the whole application rather than using DEX file only. By giving a lightweight solution this model detects malware variants and obfuscated and repacked software with an accuracy of more than 76%. In [6] comparison is done between the hash of a file section with the hash stored in a database of malware to identify whether testing PE is a malware variant or not. This technique uses ssdeep algorithm to create a hash of each section of the PE file and uses fuzzy hashes to identify the level of similarity in both files and gives an accuracy of 98.16% in malware variant detection. This proposed method gives a high false negative rate on section-level hashing that needs to be improved. In this proposal, the fuzzy hash technique will be implemented on sections of APK file. the detailed sections of APK file are shown in Figure 1.1. These sections will be used for further analysis to identify whether the file is goodware or malware.

In [22], a comprehensive and advanced approach for investigating variations in Android malware has been presented. The authors have introduced a novel fingerprinting technique called APK-DNA, which goes beyond traditional fuzzy hashing methods by capturing not only the binary code of the APK file but also its structural and semantic characteristics. This unique fingerprinting method ensures high resistance to changes in the app, making it highly effective in identifying different versions of malware. Furthermore, the researchers have utilized the APK-DNA fingerprint to develop a cutting-edge framework named ROAR, which aims at detecting Android malware through two distinct approaches: family fingerprinting and peer-matching. By leveraging this framework, not only can malware variations be identified, but the specific family to which the detected malware belongs can also be attributed. The evaluation of the ROAR framework has yielded highly promising outcomes, demonstrating exceptional accuracy in terms of both malware detection and family attribution. Recent experiments conducted in this study demonstrate that the proposed fingerprinting technique and the implemented ROAR system achieve an impressive precision rate of 95%. These impressive results highlight the effectiveness and robustness of the proposed approach, making it a valuable contribution to the field of Android malware analysis and detection.

Rather than comparing API call sequences, FUMVar [23] takes into account the similarity in API calls when comparing malware variants with their original samples. By utilizing ssdeep and the Jaccard distance, we can effectively assess the similarity and dissimilarity, respectively, between these elements. This approach enhances the ability to analyze and understand the characteristics and behavior of malware variants in a comprehensive manner.

Ibrahim et al. [24] introduce a novel approach involving static analysis and comprehensive feature extraction from Android applications. The method incorporates two newly proposed features and feeds them into a functional API deep learning model developed for the purpose. The approach is applied to a freshly curated dataset comprising 14,079 samples encompassing both malware and benign applications. The malware samples are further categorized into four distinct classes. Two key experiments are conducted using this dataset. The first experiment involves binary classification for malware detection, where samples are categorized as either malware or benign. In this scenario, the model achieves an exceptional F1-score of 99.5%, surpassing the performance of similar approaches. The second experiment extends to multi-class classification, encompassing all five classes of the dataset. Remarkably, the model still achieves a high level of accuracy, yielding an F1-score of 97% for malware detection and classification.

[25] investigates to assess the efficacy of hash values in malware detection. Initially, Deep Learning (DL) techniques are applied to analyze the surface information of Portable Executable (PE) files, in conjunction with hash values, to assess their effectiveness in identifying malware. Notably, remarkable performance gains are observed when incorporating PE surface information with impfuzzy hash values. These findings underscore the potency of DL as a po-

tent tool for static property analysis logs. Furthermore, this paper extends its research to encompass data characterized by concept drift properties—a phenomenon often encountered in dynamic malware environments. Here, impfuzzy demonstrates notable effectiveness in handling such data types. Intriguingly, an experiment is devised in which hash values and PE surface information are synergistically combined. The outcome is the attainment of significantly enhanced performance compared to using PE surface information in isolation. For instance, relying solely on PE surface information yields a classification accuracy of 0.9148. In contrast, when the PE surface information is complemented with hash values, the highest classification accuracy surges to an impressive 0.9198. These results compellingly advocate for the utility of incorporating hash values in the arsenal of techniques for the detection of previously undiscovered malware.

Choi et al. addressed challenges in malware detection using Artificial Intelligence (AI) and similarity hash-based methods [26]. In their research, an introduction is provided regarding a k-nearest-neighbor (kNN) classification approach coupled with a vantage-point (VP) tree utilizing a similarity hash. The methodology significantly enhances detection efficiency. Specifically, the research highlights a 67% reduction in detection time compared to previous methods, in addition to a noteworthy 25% increase in detection rates. Furthermore, the utilization of a VP tree in conjunction with a similarity hash results in a remarkable 20% reduction in search time for similarity hashes.

Table 2.3: Comparison Table Literature Review

NO	Technique	Dataset	File Type	Malware Families	Variant Identification	Classification Type	Limitation	Accuracy
[8]	Fuzzy Hash And Clustering Algorithm	Google Play & Third Party Applications	Android	Droid kungfu Anserver bot Cruise win	yes	Binary	Less Accuracy	70%
[6]	Section Level Fuzzy Hashing	Windows 32 EXE And DLL PE File Malware Collection	PE	Virut, Wabot, Virlock & Coinminer	Yes	Family	High rate of false Negtive	81%
[7]	Fuzzy Hash And N-Gram	Undefined	Android	Undefined	yes	Binary	Samples are incorrectly clustered when they are packed with the same packer	Undefined
[21]	Fuzzy Hash Image Processing N-Gram	-	Android	Undefined	No	Binary	-	98.5%
[27]	Malware Variant Generation Using Genatic Algorithm	The Zoo	Android	Ares Radium	Yes	Binary & family	limited malware families are used	-
[28]	Similarity Hash based scoring	From Repository Of Nettitude Ltd, UK	PE	Trojan Adware Worm Downloader	No	Binary & family	scope is limited to IoTs only	90%
[29]	Fuzzy And Import Hashing	Hybrid Analysis & Malshare	PE	Wanna cry Locky Cerber Cryptowall	Yes	Binary	no significant improvements	-
[30]	Fuzzy Hashing & YARA Rules	Hybrid Analysis & Malshare	PE	Wanna cry Locky Cerber Cryptowall	Yes	Binary & family	Undefined	Improved
[22]	Section based hashing using reverse engineering & ngram	Android Malware Genome Project	Android	Anserver Bot Droid KungFu4 Droid KungFu3 KMin	Yes	Binary & family	-	94%

[31] introduces the need for an efficient method to detect obfuscated malware in Android-based smartphones. While existing approaches for Android malware classification exist, they lack the ability to detect obfuscated malware effectively and adaptively improve their detection rules. The paper proposes an evolving hybrid neuro-fuzzy classifier (EHNFC) based on soft computing systems. This EHNFC not only detects obfuscated malware using fuzzy rules but also evolves by learning new malware detection rules to enhance its accuracy. The research modifies an evolving clustering method to incorporate an adaptive process for updating clustered permission-based features' radii and centers. This modification improves cluster convergence and generates rules better suited to input data, thereby enhancing the EHNFC's classification accuracy. Experimental results show that the proposed EHNFC outperforms

state-of-the-art obfuscated malware classification methods in terms of false negative and false positive rates (0.05) and achieves a superior accuracy of 90% compared to other neuro-fuzzy systems.

In another similar research, Windows API call sequences were used to capture the behavior of malicious applications [32]. The Detours library by Microsoft was used to hook these Win-API call sequences. To simplify the analysis, related Win-APIs were grouped into 26 categories (A...Z). The behavior of five classes of malware (Worm, Trojan-Downloader, Trojan-Spy, Trojan-Dropper, and Backdoor) was studied using 400 samples for each class, totaling 2000 samples for training. Additionally, 120 samples for each class were used for testing. Fuzzy hashing with ssdeep generated signatures, which were then matched to identify similarities in API call sequences between test and training samples. This approach yielded promising results in classifying samples into the five malware categories. N-gram analysis was also conducted to extract distinct API call sequence patterns for each category.

Li et al. explored in their research the similarity analysis of Control Flow Graph (CFG), which is a crucial technique for security tasks like malware detection and clustering [33]. Current CFG similarity methods face challenges in efficiency, accuracy, and user-friendliness. To address these issues, a new fuzzy hashing method named topology-aware hashing (TAH) is introduced. TAH works by extracting blended n-gram graphical features from CFGs, converting these features into numeric vectors (graph signatures), and then measuring graph similarity by comparing these signatures. Additionally, fuzzy hashing is used to transform numeric graph signatures into compact fuzzy hash signatures, making similarity calculations more efficient. Extensive evaluations indicate that TAH outperforms existing CFG comparison techniques in terms of both effectiveness and efficiency. To showcase TAH's practical use in security analysis, a binary similarity analysis tool based on TAH is developed and tested. The results demonstrate that this tool surpasses existing similarity analysis tools, particularly in the context of malware clustering.

A new technique called APK-DNA, inspired by fuzzy hashing, is introduced for fingerprinting Android apps, with a focus on identifying malicious ones [34]. APK-DNA computation is efficient and quick for suspicious apps. Building on the concept that very similar apps might belong to the same malware family, a cybersecurity framework named Cypider (Cyber-Spider for Android malware detection) is proposed. Cypider combines various techniques to address Android malware issues, including clustering and fingerprinting. It can detect repackaged malware (malware families) and identify new malware apps automatically and without prior knowledge. The core idea behind Cypider is to create a similarity network based on fuzzy fingerprints of the apps' static content. From this network, it extracts highly connected sub-graphs, called communities, which are likely to contain malicious apps.

[35] introduces an innovative approach for enhancing ransomware triaging, involving the integration of augmented YARA rules with fuzzy hashing techniques. Initially, the study employed ransomware triaging through a combination of fuzzy hashing, import hashing, and standard YARA rules. Specifically, it utilized three distinct fuzzy hashing methods: SSDEEP, SDHASH, and mvHASH-B, along with the import hashing method IMPHASH, and traditional YARA rules. The triaging results revealed that the fuzzy hashing methods consistently outperformed YARA rules and IMPHASH, showcasing their effectiveness in specific scenarios where YARA rules fell short. Consequently, augmented YARA rules combined with fuzzy hashing were devised, yielding superior results when compared to individual triaging techniques. Furthermore, the study assessed the performance efficiency of the fused rules in contrast to standard YARA rules. The findings indicated that combining YARA rules with the SSDEEP fuzzy hashing method could achieve nearly equivalent performance efficiency as standard YARA rules. However, it was noted that the other two fuzzy hashing methods, SDHASH and mvHASH-B, while effective,

might entail slightly higher performance overheads. The proposed fused YARA rules exhibited strong performance within the context of the WannaCry corpus and implementation. Future research will involve testing these fused rules with larger sample sizes and more diverse sets of ransomware and malware, further assessing their applicability and efficacy.

The research in [36] introduces a novel framework designed to detect malicious Android applications, specifically focusing on those containing repackaged malicious code. The framework employs a meticulous process of feature extraction from Android applications, utilizing their source code. These extracted features are subsequently compared to existing malware signatures, allowing for the identification of repackaged malicious Android applications. To validate the framework's efficacy, extensive experiments were conducted using a dataset comprising 3,490 samples of Android-based malware, spanning 21 distinct malware families. The research established a crucial threshold for categorizing malware, employing fuzzy logic. According to this threshold, if the fuzzy comparison match exceeds 40%, the application is deemed malicious. In cases where the match falls between 10% and 40%, the application is labeled as suspicious; otherwise, it is considered benign. Remarkably, the results obtained from the proposed framework showcase its effectiveness. Specifically, it successfully identifies approximately 74% of repackaged malware instances, outperforming other similar approaches in this critical domain of cybersecurity. This research offers a promising avenue for enhancing Android malware detection by targeting the insidious issue of repackaged malicious code, thereby fortifying the security of Android users.

Sarantinos et al. [37] conducted research centered on evaluating the efficacy and efficiency of fuzzy hashing algorithms within the domain of Malware Analysis. More specifically, it seeks to highlight the advantages of incorporating fuzzy hashing techniques like ssdeep, sdhash, mvHash, and mrsh v2 in the identification of similarities among malware instances. These results will be contrasted with the traditional cryptographic hash functions, such as MD5, SHA-1, and SHA-256, to provide a comprehensive assessment of their capabilities. Furthermore, this study underlines the strengths and weaknesses inherent in both fuzzy and cryptographic hashing approaches, along with their practical utilization in real-world applications. The implementation of fuzzy hashing emerges as notably successful in pinpointing and correlating similarities among various malware, including emerging strains within distinct families. This success critically relies on two pivotal factors: the selection of the appropriate fuzzy (SPH) algorithm for similarity detection and the availability of a hash sample extracted from one or more malware specimens. Collectively, these elements empower researchers to harness fuzzy hashing as a potent tool for malware analysis, shedding light on its potential significance in enhancing cybersecurity by bolstering malware detection and facilitating the identification of emerging threats.

In the landscape of recent cyber threats, there is a noticeable trend where attackers deploy malware variants that undergo significant updates to their existing functionalities while introducing new features. These modern malware strains are not only functionally enhanced but also exhibit improved stealth attributes, employing tactics like obfuscation, encryption, and adaptive behavior adjustments based on their runtime environment. However, the availability of skilled malware analysts is limited, which poses a challenge in effectively combating the ever-growing volume of malware instances. Consequently, there is a growing interest in developing methods that can reduce the costly manual analysis efforts required to address this proliferation of malicious software. [38] introduces an innovative approach that integrates dynamic traffic analysis with static program analysis to tackle this challenge. Similar to traditional techniques, the dynamic analysis segment focuses on feature extraction, clustering, and labeling to distill traffic data into a sequence of characters. Meanwhile, the static analysis component leverages Fuzzy Hashing, a method adept at effectively representing identical partial sections within malware programs. Three integration patterns are evaluated: prioritizing results from dynamic analysis, prioritizing those from static analysis, and utilizing the mean of both analyses. Through extensive experimentation involving 340 malware samples and their associated traffic data, the proposed method demonstrates an impressive ability to correctly identify 61.1% of the malware instances, showcasing its potential in automating and enhancing the malware analysis process.

The study [39] delves into an extensive analysis of three prominent Android malware datasets, aiming to quantify the prevalence of repackaged malware using package name-based similarity assessments. The datasets examined include 5,560 apps from Drebin, 24,533 from AMD, and a staggering 695,470 from AndroZoo. Results reveal alarming statistics, with 52.3% of Drebin apps, 29.8% of AMD apps, and 42.3% of AndroZoo apps identified as repackaged malware. Furthermore, this research introduces AndroMalPack, an Android malware detection system trained on clone-free datasets and fine-tuned using Nature-inspired algorithms. Despite being trained on reduced datasets, AndroMalPack demonstrates exceptional detection accuracy of up to 98.2% and minimal false-positive rates. The study also provides a dataset containing cloned apps from Drebin, AMD, and AndroZoo, aimed at fostering research in repackaged malware analysis.

The practice of assigning family labels to malicious Android apps has long been used to group apps with similar behavior patterns. However, recent studies have unveiled a significant inconsistency in this approach. Apps bearing the same family label may exhibit distinct behavior, with variations in functionality. Conversely, apps labeled under different families might display remarkably similar behavior. To address this issue, the paper introduces AndrEnsemble [40], a novel characterization system for Android malware families. AndrEnsemble relies on ensembles of sensitive API calls extracted from aggregated call graphs representing different malware families. This method offers several advantages compared to existing approaches, including a superior reduction ratio in relation to original call graphs, resilience against transformation attacks, and adaptability for application at various granularity levels. Through experimental validation and examination of specific use cases, including mobile ransomware, SMS Trojans, and banking Trojans, noteworthy findings emerged. Firstly, these malware types do not consistently employ multiple sensitive API calls in unison to execute malicious operations. Secondly, SMS Trojans tend to possess larger ensembles of API calls compared to other types. Lastly, instances were identified where different malware families shared identical API call ensembles despite being categorized separately. This research highlights the complexity and variability in Android malware behavior, emphasizing the need for more precise and nuanced classification methods.

In recent times, there has been a notable increase in state-sponsored malware activities. Advanced Persistent Threat groups (APTs) have engaged in covert cyber conflicts, often avoiding consequences due to the secretive nature of these operations. To address this issue and potentially impose sanctions, it's crucial to attribute the malware used in these attacks. Malware attribution is a pivotal step in understanding the methods employed by APT attackers, especially their exploitation of known vulnerabilities to gain access to target networks. Previous efforts in automated attack attribution have utilized behavior reports from sandbox environments as inputs for machine learning algorithms. While this approach is generally reliable, it has limitations. Some APT files can detect sandbox environments and alter their behavior, leading to erroneous or inconclusive attributions. Therefore, there's a need for an alternative technique to extract features for attack attribution. The research by Kida et al. [41] introduces an innovative framework for lightweight and automated attack attribution. It leverages fuzzy hashes as natural language input for machine learning classifiers, enabling the attribution of attacks. Experimental results indicate that this framework achieves an average accuracy of 89% and an F1-score of 87.5% for both country and APT group classification. Furthermore, the proposed approach offers a quicker and more efficient method for attack attribution, enhances in-depth analysis of malware samples, and delivers competitive performance compared to state-of-the-art dynamic analysis attribution tools.

This study in [42] delves into the application of similarity digests, with a focus on TLSH, in security tasks such as malware identification. TLSH, known for its robustness against evasion tactics compared to counterparts like ssdeep and sdhash, emerges as a powerful tool. The research addresses the challenges of searching and clustering vast amounts of malware and goodware data efficiently. It introduces innovative techniques that enable rapid search and clustering of TLSH hash digests. Leveraging efficient nearest-neighbor search methods and a tree-based index, the proposed threshold-based Hierarchical Agglomerative Clustering (HAC-T) algorithm exhibits remarkable scalability. Empirical evaluations against standard clustering methods highlight its superior performance. Notably, it achieves high purity scores, ranging from 0.97 to 0.98, using data from VirusTotal, showcasing its effectiveness in practice. This approach promises to significantly enhance the speed and precision of malware analysis and clustering tasks, making it a valuable addition to security operations.

Fake IoT applications pose an escalating threat to IoT security due to their low development costs and high profitability. MSimDroid [43] introduces a pioneering method for detecting fake IoT apps based on multidimensional similarity. MSimDroid concentrates on scrutinizing distribution channels, such as app markets, using a multifaceted approach encompassing whole app, resource, and code similarity, supported by a specialized algorithm. A joint strategy optimally balances accuracy and efficiency. Experiments confirm MSimDroid's effectiveness, achieving over 99.31% accuracy on a ground-truth dataset and 97.43% in real-world scenarios. Analysis of IoT apps from various markets reveals that about 14.66% are fake, with malicious behavior detected in around 0.58% of IoT apps.

Securing cloud platforms and ensuring their resilience is a pressing challenge, given the multitude of diverse applications sharing resources. Detecting malware and threats within the cloud is crucial. While machine learning-based malware analysis has been explored, it often suffers from computational complexity and limited accuracy. Kumar et al. [44] address these limitations and propose a novel malware detection system by combining clustering and trend micro locality sensitive hashing (TLSH). The system leverages the Cuckoo sandbox for dynamic analysis reports, executing files within an isolated environment. The feature extraction algorithm is applied to extract essential data from Cuckoo sandbox malware reports. Key features are then selected through methods like principal component analysis (PCA), random forest, and Chi-square feature selection. Experimental results are obtained for both clustering and non-clustering approaches using Decision Tree, Random Forest, and Logistic Regression classifiers. This novel model exhibits enhanced classification accuracy and a reduced false positive rate (FPR) compared to existing methods, all while significantly reducing computational costs.

The proliferation of Android usage in recent years has brought forth a corresponding increase in security threats, primarily stemming from code reuse in various applications. In response, Akram et al. [45] introduce DroidSD, a clone detection tool tailored for Android applications. DroidSD is designed to identify different types of code clones within APK source code effectively. This tool can accurately detect Type-1, Type-2, and Type-3 (near-miss) clones at the source code level, marking a substantial improvement over previous Android similarity detection techniques. DroidSD is capable of discerning full and partial similarities among applications and demonstrates impressive Recall and Precision rates when tested against real-world datasets, solidifying its efficacy in Android app code clone detection.

The paper [46] introduces DroidMD, a scalable self-improvement tool designed to detect malicious Android applications at the source code level. This tool aims to address the increasing threat of Android malware that poses risks to users and the Android marketplace. DroidMD utilizes an auto-optimization approach for its signature set, enhancing its ability to identify malware in applications. The researchers conducted evaluations on a substantial dataset, consisting of approximately 30,000 applications, including 27,000 benign and 3,670 malware applications. DroidMD excels in detecting malware at both partial and full levels within applications. Importantly, it analyzes only the application's code, increasing its reliability. The results of the evaluation demonstrate DroidMD's effectiveness, achieving a high accuracy rate of 95.5%. This research contributes to improving the security of Android markets by providing a robust tool for identifying and mitigating the threat of malware-infected applications. Table 2.3 presented in the literature review showcases research endeavors employing cryptographic and similarity hashes for the detection of Android malware and its variants. This table offers a summary of studies concentrating on binary and family classification, as well as variant detection, while also shedding light on the limitations identified within these research works.

CHAPTER 3

METHODOLOGY

This chapter covers the details and practical implementation of how fuzzy or similarity hashes are efficient to perform variant detection and family classification. Through a systematic and structured approach, we unveil the inner workings of our methodology, empowering the reader with insights into our research process. The problem at hand revolves around the detection and categorization of Android malware and its variants, a critical issue in the realm of mobile device security. With the proliferation of Android applications, the risk of encountering malicious software has escalated significantly. Traditional signature-based detection methods are no longer sufficient to combat the evolving landscape of Android malware, which frequently mutates to evade detection. Thus, a pressing need exists for a sophisticated approach that leverages the power of hash values and similarity analysis to identify Android malware and variants effectively.In this paper we introduced an approach to detect Android malware using static analysis. The method leverages the extraction of essential features from Android applications to identify potential malware variants. Similarity hashes are calculated and further processed to perform binary and family classification of testing apk.

The block diagram displayed in Figure3.1 elucidates the underlying methodology upon which our framework operates. This visual representation details the process of hash calculation, storage, and subsequent comparison following the decompilation of APK files and the extraction of features.

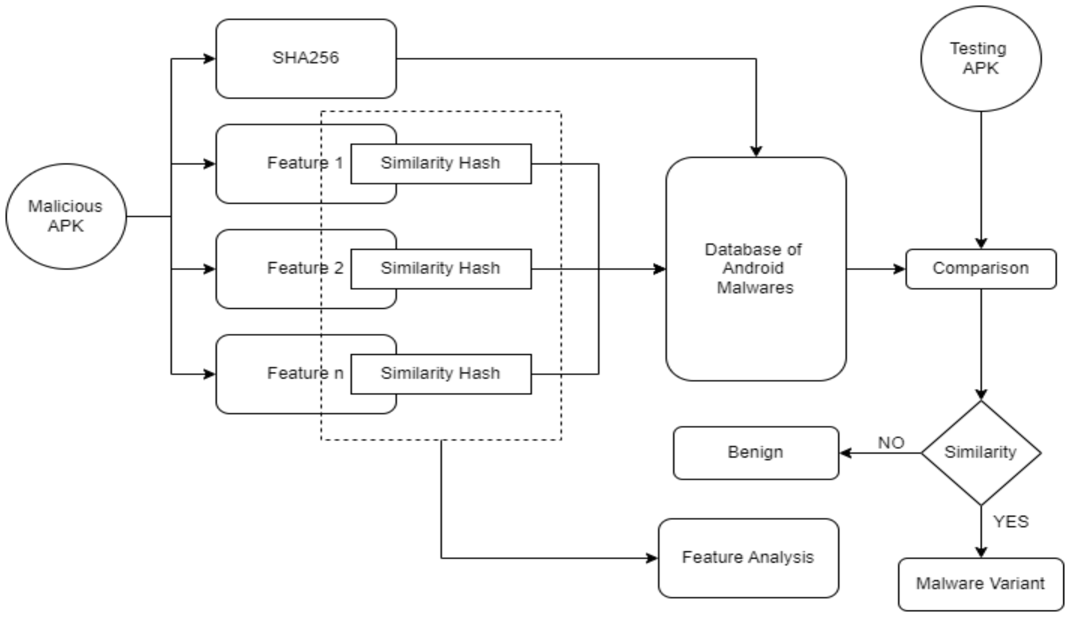

Figure 3.1: Working Model of Proposed Framework for Android Malware and Variant Detection

3.1 Dataset

The dataset utilized in our research was sourced from Drebin [2]. This dataset is extensive, containing a substantial number of entries, including 123,453 benign and 5,560 malicious Android applications. For the purposes of our research, we carefully narrowed down the dataset to select 500 benign applications and 1500 malicious applications, ensuring a representative sample for our study.

It includes static attributes such as api-call, resource names, features, methods, and manifest permissions, etc. that are extracted from the APK files of dataset.

3.2 Experimental environment

The research experiments were conducted using custom Python application programs running on an Ubuntu 21.10 LTS operating system. These applications were tailored to meet the specific research needs efficiently. The research environment was hosted on a VMware Workstation 16 virtual machine, equipped with 8 GB of RAM 4 core processors with hyperthreading, and a 1.5 TB disk. This setup provided the necessary computational resources for conducting the experiments effectively. provides a comprehensive breakdown of the external libraries and utilities that were incorporated into the research framework. Table 3.1 provides a list of external libraries and utilities that have been used in the development of the experimental environment and the execution of experiments.

Table 3.1: External Libraries & Utilities

Name	**Description**
apktool	A tool used for APK reverse engineering
dex2jar	to convert the classes. dex file to classes. jar or vice versa
ssdeep	A tool used to generate similarity hashes of files
Virus Total	Malware Labelling tool
Malware Bazaar	Malware Labelling tool
pymongo	Database Library

3.3 Malware Categories and Family Classification

In our study, we have chosen to work with six most popular Android malware families. Therefore, we have divided our dataset into six primary categories: trojan, adware, spyware, virus, hacktool and downloader, with each category containing various malware families. Table 3.2 offers a comprehensive overview of the malware families that have been integrated into our research.

The number of malware samples within each family varies randomly, creating a diverse dataset.

Table 3.2: Malware Categories and Included Families

Malware Category	**Malware Families**
Trojan	droidrooter, smsflood, rooter, fakeplayer, smssend, fakemobi, droidsms, droidkungfu, opfake, fakeinst, ginmaster, gmaster, gapev, kungfu, andr, legana, plankton, plangton
Adware	plankton, plangton, leadbolt, kungfu, droidkungfu, andr, startapp, ginmaster, gingermaster, gmaster, apperhand
Spyware	droidsnake, mobilespy, flexispy, fspy, smsrep, replicator, gpsspy, smsreplicator, gpspy, tapesnake
Virus	Ginmaster, gingermaster, gmaster
Downloader	ddlight, lightdd, dordrae, ozotshielder, kmin
Hacktool	hacktool, lotoor

To ensure the accuracy and consistency of our categorization, we adopted the malware classifications and family information from Malware Bazaar and VirusTotal. This approach ensures that our dataset aligns with widely recognized malware taxonomies, enhancing its reliability for research and analysis. To facilitate efficient data management and retrieval, we stored this categorized data, along with the SHA256 hash of each APK file, into a centralized database. This structured database allows us to organize, query, and analyze the dataset effectively.

3.4 Selection of Fuzzy Hashes

We opted for ssdeep in our research due to its remarkable efficiency and accuracy in identifying similarities when compared to alternative hashing methods. A research conducted by Baba et al. evaluates the performance of nine different fuzzy hashing techniques. The results shown in Table 3.3 determines which hash function outperforms the rest in terms of efficiency.

Additionally, the overview presented by Botacin et al. [47] highlights that among the recently published works, ssdeep emerges as the most widely adopted similarity hash function for detecting similarities between two files. This preference is indicative of its reliability as an approach. Moreover, the comparative analysis presented by Naik et al. [48] conclusively demonstrates that the SSDEEP fuzzy hashing method outperforms the other two methods, SDHASH and mvHASH-B, in the context of similarity detection.

Table 3.3: Performance of Fuzzy hashes [1]

	Accuracy	**Precision**	**Recall**
impfuzzy	0.8871	0.8869	0.8874
ssdeep	0.6993	0.6996	0.6979
TLSH	0.6187	0.6187	0.6189
imphash	0.7284	0.7284	0.7277
totalhash	0.6705	0.6704	0.6704
endgame	0.6701	0.6698	0.6680
anymaster	0.6959	0.6959	0.6959
crits	0.6238	0.6237	0.6242
pehashng	0.6836	0.6836	0.6833

3.5 Feature extraction & selection

Optimizing accuracy in Android malware detection necessitates a crucial step: feature selection post-feature extraction. Our extensive research and detailed analysis on components of Android applications, has led us to pinpoint the most effective features for this purpose.

3.5.1 APK file Size

Android malware often disguises itself in compact applications to evade detection and facilitate quick downloads. It is observed that malware samples tend to have considerably smaller file sizes compared to benign apps. Typically, benign applications are approximately three times, or even more, the size of their malware counterparts [49]. By monitoring and comparing the sizes of Android applications extracted from APK files, we can identify unusually small apps, which may indicate the presence of malware.

3.5.2 Permissions

Malicious applications frequently request excessive or suspicious permissions to gain unauthorized access to sensitive data or device functions. Analyzing app permissions extracted from the Manifest.xml file is critical. It helps identify apps that seek permissions beyond what is necessary for their legitimate functionality. Fuzzy hashes can be employed to identify similarities in permission patterns among apps. This aids in detecting apps with suspicious permission profiles.

3.5.3 Fuzzy Hash

Fuzzy hashing is employed to identify similarities between files, even when they undergo slight modifications. Comparing fuzzy hash values computed from the content of Android apps can help detect variations of known malware and previously unseen threats.

3.5.4 Features

The AndroidManifest.xml file contains essential information about the app's components and their functionalities. Malware may hide malicious behaviors in the manifest. Extracting manifest features aids in understanding an app's declared functionalities. Anomalies or discrepancies in the manifest can indicate malicious intent. Fuzzy hashing can highlight these discrepancies or similarities in manifest features among apps, assisting in the identification of hidden malicious intents.

3.5.5 Activities

Activities can be extracted from the AndroidManifest.xml file within the APK, where they are declared along with their associated attributes. Activities and actions define how an app interacts with users and other components. Malicious apps might exploit these components to carry out harmful actions. Monitoring activity and action usage helps pinpoint apps attempting to perform malicious activities through deceptive interfaces or unexpected actions. Fuzzy hashing can help identify apps that share similar activity and action patterns, indicating potential malicious behavior.

3.5.6 Services

Services are declared in the AndroidManifest.xml file, similar to activities. Examining this file reveals the services defined within the APK. Services are background processes in Android that perform tasks independently of the user interface. They are often used for long-running operations such as downloading files or playing music in the background. Services are often used by malware to run malicious operations discreetly. Analyzing services can help identify hidden, potentially harmful background activities.

3.5.7 Intent

Intents are typically defined in Java code and XML files. By examining an app's source code, you can identify the intents it uses or registers. Intents are a messaging mechanism used for communication between components within an app or between different apps. They can request actions or pass data between components. Suspicious or malicious intents can be monitored to identify apps trying to communicate with or manipulate other apps in unauthorized ways, which may indicate malware activity.

3.5.8 Content Providers

Content providers can be extracted from the AndroidManifest.xml file, which lists the providers and their permissions. They are also identified by their Uniform Resource Identifier (URI) in code. Content providers manage access to structured data, allowing data sharing between different apps. They provide a structured interface to interact with data stores, like databases or files. Examining content providers is crucial for identifying unauthorized data access, which could be indicative of data theft or misuse by malware.

3.5.9 Broadcast Receivers

Broadcast receivers are declared in the AndroidManifest.xml file and associated with specific events or messages they can receive and respond to. Broadcast receivers listen for system-wide events or messages, such as device booting or incoming SMS messages. They can respond to these events or trigger specific actions in response. Malicious apps may use broadcast receivers to intercept sensitive data or execute unauthorized actions when certain events occur. Monitoring these receivers is crucial for detecting suspicious behavior.

3.5.10 Source Class Package and Method Name

It can be extracted from classes.dex file of an APK. The source class package provides insights into the origin of code within the app. It reveals potential third-party or obfuscated code. Examining the source class package assists in identifying suspicious code origins. Malware often incorporates code from untrusted sources. fuzzy hashes can assist in comparing source class packages and method names across apps, making it easier to spot code reuse and obfuscation techniques.

3.5.11 API Calls

API calls can be extracted by decompiling APK using tools like 'apktool' and 'dex2jar'. These are a set of protocols and tools that allow different software applications to communicate with each other. In simpler terms, API calls are like a language that software uses to interact with other software, enabling them to perform various tasks and functions. Malware belonging to the same family often exhibits common imported API functions, allowing for their recognition through similarity hashes like impfuzzy hash.

3.5.12 Certificate Owner

Certificates used to sign apps provide information about their legitimacy. Applications with similar functionalities can use similar certificates. Fuzzy hashes can be used to identify malware variants by spotting similar certificates used by parent malware.

3.6 Data collection and formation of Hashes

Data collection involved the utilization of two Python programs. The initial program was designed to create and archive ssdeep similarity hashes corresponding to the extracted features of identified APK files. The subsequent program was responsible for evaluating the feature hashes of a selected APK file by comparing them to the entries stored within the database, thereby deducing potential family associations.

3.7 Variant Identification & Family Classification

Traditional cryptographic hash functions, such as MD5 and SHA1, are celebrated for their security applications. However, they exhibit a significant drawback: even the slightest alteration in input data, differing by just one bit, yields entirely distinct hash values. While this property is invaluable for data integrity and security verification, it falls short when handling samples that closely resemble known malware.

This is where similarity hash functions come into play. In our research, we categorized our dataset into various malware categories, as previously mentioned in Table 3.2. Unlike some other studies [22], where the entire file or folder's fuzzy hash is used for malware detection, we took a different approach. We gathered fuzzy hashes from features extracted specifically from APK files to assess accuracy in both malware categorization and family classification.

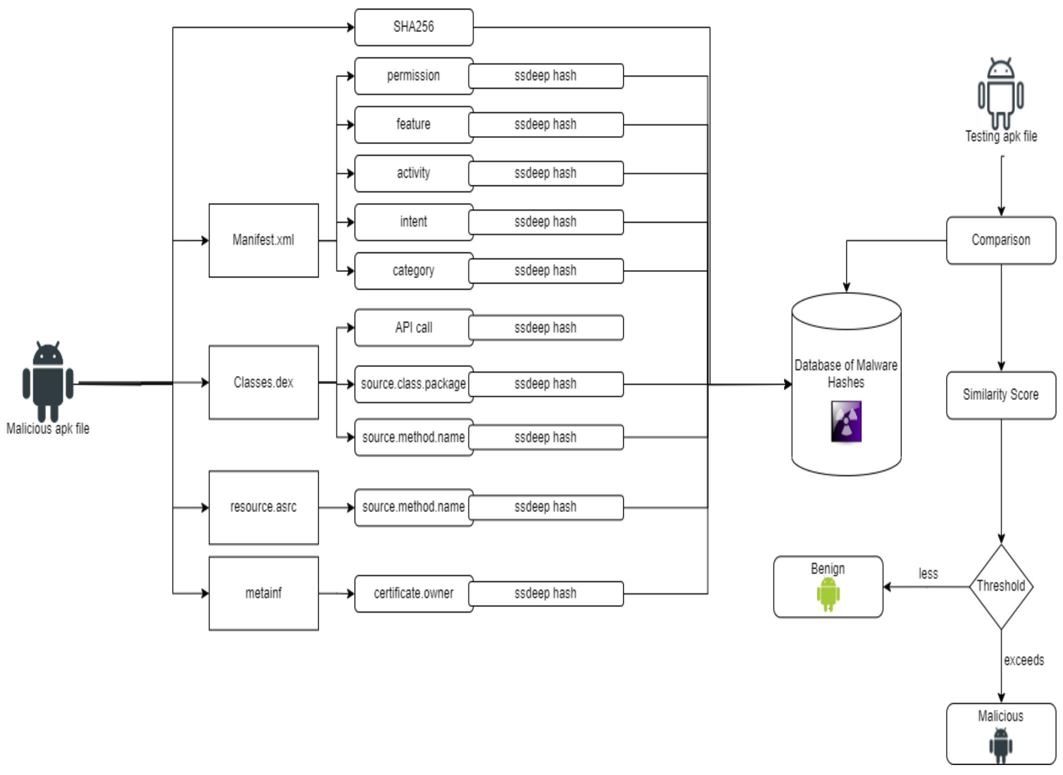

Figure 3.2: Foundational Architecture of Proposed Framework for Android Malware and Variant Detection

The results of our experiments across different malware categories are presented in Tables 4.2, 4.3, 4.4, and 4.5 in the results section. This approach allowed us to focus on the specific attributes that matter most for accurate malware categorization and family classification, offering a nuanced and effective method for enhancing Android malware detection.

Algorithm 1 Calculating & Storing ssdeep Hash to DB

```
0: procedure FUNC_HASH($input, db$)
0:     hash ← calculateSsdeepHash($input$)
0:     SaveHashToDB($hash, db$)
0: end procedure
0: function FUNC_HASH($input$)
0:     hash ← ssdeep.hash($input$)
0:     return hash
0: end function
0: procedure SAVEHASHTODB($hash, db$)
0:     ConnectToDB($db$)
0:     InsertHashIntoDB($hash$)
0:     CloseDBConnection()
0: end procedure
   =0
```

3.8 Detailed Explanation of proposed method

Our primary objective was to establish a comprehensive database enriched with cryptographic and similarity hashes, derived from the extracted static features of well-known malware APK files. Figure 3.2 provides an illustrative overview of the fundamental operations of our proposed framework designed for the detection of Android malware and its variants. To accomplish this foundational task, we initiated the development of a Python script, designed to obtain SHA256 hashes. These hashes not only helped in the identification of known malicious applications, but also in linking features and sub-features with their parent APK files.

To address the complex task of variant detection, we integrated a robust tool named ssdeep into our program. Ssdeep is renowned for its proficiency in calculating fuzzy or similarity hashes, a crucial technique that would enable us to detect Android malware variants with greater similarity. This inclusion helped our methodology to not only identify malware but also distinguish between different variants of the same malicious application. The pseudocode, as depicted in Algorithm 1, elucidates the process by which the calculation is performed to obtain fuzzy hashes and how stored in the database.

Algorithm 2 Calculate Similarity Index

```
0: for each record $d$ in $db$ do
0:   $reference\_value \leftarrow GetSHA256valuefrominput\_apk$
0:   for each $feature$ in $d1$ do
0:     if $key$ not in $keys\_to\_skip$ then
0:       $similarity\_index \leftarrow$ Compare the values of features in $d1$ and $d2$
   using $ssdeep$
0:       OUTPUT $feature + " : " + match\_score$
0:       $Net\_similarity\_index \leftarrow$ add similarity_indexes with the same
   $reference\_value$
0:       if $similarity\_index < 40$ then
0:         OUTPUT "APK with SHA256" $+reference\_value+$ "is benign"
0:       else
0:         OUTPUT "APK with SHA256" $+reference\_value+$ "is Mali-
   cious"
0:       end if
0:     end if
0:   end for
0: end for=0
```

In another process shown in Algorithm 2 we initiated by obtaining the SHA256 hash of the testing APK, preserving it for later use. Subsequently, the same APK underwent decompilation using tools like apktool and dex2jar, enabling us to extract the necessary features. We then computed fuzzy hashes for these extracted features, facilitating comparison with the hashes stored in our reference database.

A visual representation of the same process can be derived by referring to Figure3.3, which illustrates a block diagram depicting the calculation of fuzzy hashes and the subsequent computation of similarity indices. This figure provides an indepth overview of the entire process, enhancing the understanding of how fuzzy hashes are calculated and similarity indices are determined.

3.8.1 Setting Threshold Value

To set threshold values, our methodology drew insights from existing research conducted by Shiel et al. [6], Lee et al. [7], and Ali et al. [36]. The similarity index generated by ssdeep spans from 0 to 100, where 0 signifies dissimilarity, and 100 indicates similarity between files. To categorize the testing APK, we instituted a threshold. If the similarity index yield by ssdeep falls below the threshold value, the APK is classified as benign. Conversely, if it surpasses the threshold value, the APK is identified as suspicious or potentially malicious. Existing research presented by Lee et al. [7]and Ali et al. [36] states that threshold values typically range between 0-10 for benign, 10-40 for suspicious, and 40 or above for malicious applications. Taking into account the findings from prior research, we established a threshold at the value of 40. This thresholding process ensures precise classification of Android applications, enhancing our malware detection and variant identification capabilities.

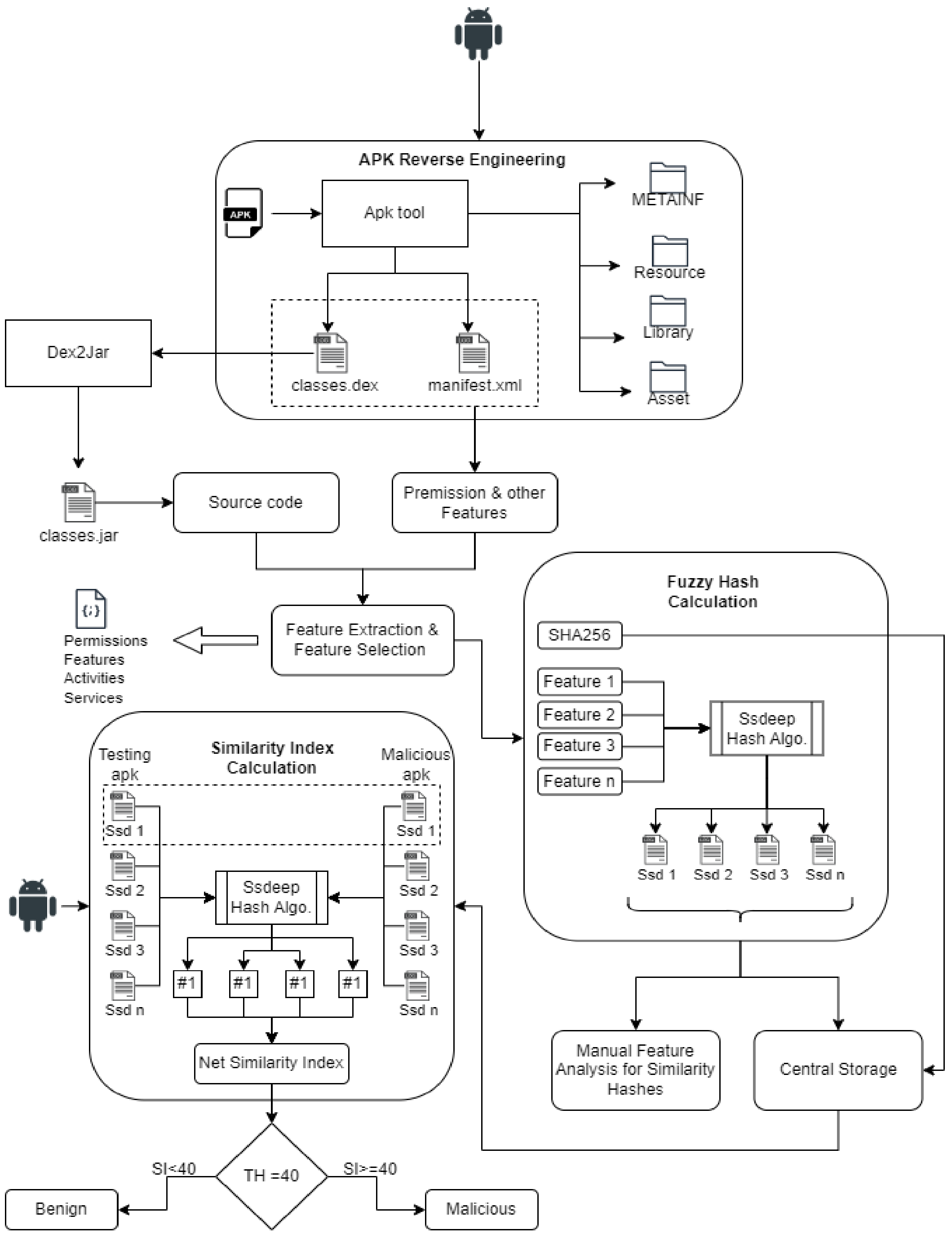

Figure 3.3: Detailed Architecture of Proposed Framework

3.8.2 Performance metrics

In the context of malware detection, there are four possible outcomes named as True Positive/False Positive, True Negative/False Negative. The detail of these outcomes has been described in Table3.4

Table 3.4: Possible Outcomes in Malware Detection domain

Output	**Description**
True Positive (TP)	A true positive occurs when the detection system correctly identifies a sample as malware.
True Negative (TN)	A true negative happens when the detection system correctly identifies a sample as benign (not malware).
False Positive (FP)	A false positive takes place when the detection system incorrectly flags a sample as malware when it is actually benign.
False Negative (FN)	A false negative occurs when the detection system mistakenly categorizes a sample as benign when it is actually malware.

In 2015, Upchurch and Zhou introduced a framework called "Variant" [50] for testing programs that detect malware variants using similarity comparisons. They also introduced key performance metrics that depend upon true/false outcomes that are discussed above. These key metrics are:

Precision: Measures the accuracy of correctly classified samples. It is the ratio of correct positive to all positive (correct & incorrect) diagnoses.

$$Precision = TP/(TP + FP) \tag{3.1}$$

Recall: Evaluate the algorithm's ability to find relevant instances. It is the ratio of correct positive diagnoses to correct positive and incorrect negative diagnoses (when the inputs are all malware)

$$Recall = TP/(TP + FN) \tag{3.2}$$

F-Measure: A harmonic mean that considers both Precision and Recall, providing an overall measure of performance.

$$F - Measure = 2 * (Precision * Recall)/(Precision + Recall) \tag{3.3}$$

CHAPTER 4

ANALYSIS & RESULTS

This chapter gives the evaluation of this work. We compute and evaluate recall, precision, and accuracy metrics to gauge the effectiveness of our framework and finally, we discuss the experimental results for the proposed approach. This empirical assessment forms the basis for drawing meaningful conclusions regarding the capabilities of our malware detection approach.

4.0.1 Testing Setup and Experimentations

The experiments were designed to assess the detection system's performance. To achieve this, we partitioned the malicious dataset, consisting of 2000 files, into two segments: 80% (1600 files) were utilized to build a database with their corresponding hashes, while the remaining 20% (400 malicious) were reserved for testing purposes. In our subsequent experiments, we introduced an additional dataset comprising 500 benign APK files. This addition served the purpose of assessing the accuracy of our binary classification model, allowing us to evaluate its performance comprehensively.

In the database creation phase, we employed a hashing mechanism to generate fuzzy hashes for features of the 1600 malicious files, storing these hashes with SHA256 of APK in a MongoDB database. These fuzzy hashes serve as reference points for comparison during the testing phase.

Table 4.1: Similarity Index & Feature Analysis for Binary Classification

	Api call	**Perm.**	**Url**	**Provider**	**Feature**	**Intent**	**Activity**	**Call**	**Srvc rcvr**	**Real perm.**	**Net SI**	**SI w/o PFI**
1	0	0	0	100	12	100	0	0	0	30	24.2%	3%
2	0	0	0	100	26	40	0	0	0	0	16.6%	0%
3	**54**	**19**	**100**	**100**	**26**	**100**	**0**	**0**	**100**	**30**	**52.9%**	**30.3%**
4	0	0	0	0	26	36	0	0	0	0	62%	0%
5	0	0	0	100	22	100	0	0	0	0	22.2%	0%
6	0	0	0	0	26	0	0	0	100	0	12.6%	10%
7	0	0	0	100	26	40	0	0	0	0	16.6%	0%
8	0	0	0	0	21	40	0	0	0	0	6.1%	0%
9	0	0	100	100	21	40	0	0	0	0	26.1%	10%
10	0	0	0	0	26	40	0	0	0	0	6.6%	0%

During testing, we subjected individual APK files to the detection system. To ensure a robust assessment, we configured the system with a base threshold value of 40. This threshold value plays a critical role in determining the outcomes of the testing phase.

Table 4.2: Similarity Index for Variant Identification & Family Calssification: Spyware

	Api call	**Perm.**	**Url**	**Provider**	**Feature**	**Intent**	**Activity**	**Call**	**Srvc rcvr**	**Real perm.**	**Net SI**
1	0	0	0	100	62	57	0	0	0	40	25.9%
2	**100**	**100**	**0**	**100**	**54**	**100**	**0**	**100**	**100**	**36**	**69%**
3	0	50	0	100	12	62	0	0	0	0	32.4%
4	0	0	0	0	62	620	0	0	0	11	13.5%
5	0	0	0	100	24	40	0	0	0	0	16.4%
6	0	0	0	0	62	52	0	0	0	40	15.4%
7	0	31	0	100	12	40	0	0	0	30	21.3%
8	35	35	0	0	24	20	0	0	0	0	11.4%
9	32	0	0	100	12	40	0	0	0	0	18.4%
10	29	50	0	0	31	40	0	0	0	32	18.2%

The recorded outcomes of the testing phase are categorized as follows:

(a) True Positive: This outcome signifies that the detection tool accurately predicted the malware family to which the input APK file belonged. In other words, it correctly identified the family of the input file.

(b) False Negative: In contrast, a false negative result indicates that the tool erroneously predicted a different malware family than the actual one. It incorrectly attributed the input APK file to a different family.

(c) True Negative: When the detection tool accurately predicted that the input APK file did not belong to the family being examined, it falls into the category of true negatives. It correctly identified files that were not part of the family under scrutiny.

(d) False Positive: Conversely, a false positive result suggests that the tool made an incorrect prediction by associating the input APK file with the family being analyzed when, in reality, it did not belong to that family.

Table 4.3: Similarity Index for Variant Identification & Family Calssification: Adware

	Api call	**Perm.**	**Url**	**Provider**	**Feature**	**Intent**	**Activity**	**Call**	**Srvc rcvr**	**Real perm.**	**Net SI**
1	0	0	0	100	12	40	0	35	0	0	18.7%
2	38	0	0	0	21	54	0	33	0	0	13.6%
3	0	0	0	100	21	63	0	0	0	0	18.4%
4	0	0	0	0	31	50	0	35	0	0	11.6%
5	0	0	0	0	21	13	0	0	0	0	3.4%
6	0	0	0	0	57	63	0	0	0	0	12%
7	36	0	0	100	12	64	0	0	0	0	21.2%
8	0	0	0	100	19	40	0	27	0	0	18.6%
9	**85**	**63**	**0**	**100**	**62**	**44**	**0**	**25**	**68**	**86**	**53.3%**
10	**85**	**72**	**74**	**100**	**62**	**100**	**100**	**44**	**80**	**93**	**81%**

Initially, we calculated the Similarity Index of benign applications with malicious applications to identify the performance of our framework. After manual feature analysis of static attributes of APK files, we also calculated the net Similarity Index excluding features that caused more false positives. The experiment results are presented and analyzed in a Tables 4.1, 4.2, 4.3, 4.4, 4.5. The data contained within these tables provide valuable insights into the computation of True Positives (TP), False Positives (FP), True Negatives (TN), and False Negatives (FN). In our analysis, we computed the net similarity index of an application both with and without the inclusion of columns labeled as provider, feature, and intent. Our findings revealed that these specific features significantly contributed to a higher false positive rate, introducing unwanted noise in malware detection. Consequently, the exclusion of these features resulted in clearer and more precise outcomes. Table 4.1 exhibits similarity indices for benign applications in relation to various malicious samples along with feature analysis.

Table 4.4: Similarity Index for Variant Identification & Family Calssification: Trojan

	Api call	**Perm.**	**Url**	**Provider**	**Feature**	**Intent**	**Activity**	**Call**	**Srvc rcvr**	**Real perm.**	**Net SI**
1	0	0	100	0	2	40	0	0	0	0	16.4%
2	0	53	0	17	24	64	0	0	0	0	13.8%
3	0	0	0	0	12	40	0	0	0	11	6.3%
4	0	0	0	0	31	100	0	0	0	0	13.1%
5	**100**	**83**	**100**	**100**	**100**	**100**	**0**	**100**	**0**	**100**	**78.3%**
6	0	0	0	0	54	62	0	0	0	11	12.7%
7	0	0	0	0	46	64	0	0	0	0	11%
8	0	0	0	0	57	63	0	0	0	0	12%
9	**100**	**100**	**100**	**100**	**100**	**100**	**100**	**100**	**100**	**100**	**100%**
10	**100**	**100**	**100**	**100**	**100**	**100**	**100**	**100**	**100**	**100**	**100%**

On the other hand, we conducted an analysis of randomly selected malicious samples from all considered categories to examine their similarity with malware hashes stored in the database. During comparison, the sample generates a net similarity score against each malware hash. If this score equaled or exceeded our set threshold value (in this case, 40), we categorized the testing sample as belonging to a particular malware family. Results are documented in Tables 4.2, 4.3, 4.4, and 4.5, providing a structured presentation of our research findings. For a more visual representation, Figures 4.1, 4.2, 4.3, and 4.4 are used to graphically depict the data from these tables, enhancing the clarity of our results. Subsequently, we computed critical performance metrics, including precision, recall, and F1 Score. Moreover, we assessed the overall accuracy of our framework.

Table 4.5: Similarity Index for Variant Identification & Family Calssification: Hacktool

	Api call	**Perm.**	**Url**	**Provider**	**Feature**	**Intent**	**Activity**	**Call**	**Srvc rcvr**	**Real perm.**	**Net SI**
1	39	0	0	100	12	100	0	38	0	30	31.9%
2	0	50	0	100	31	40	0	0	0	32	25.3%
3	**100**	**46**	**100**	**100**	**100**	**100**	**100**	**48**	**100**	**100**	**89.4%**
4	0	0	100	100	0	0	0	0	100	0	30%
5	0	0	0	100	31	40	0	54	0	0	12.5%
6	**46**	**100**	**100**	**0**	**100**	**100**	**100**	**100**	**100**	**100**	**84.6%**
7	0	0	0	100	31	40	0	44	0	0	21.5%
8	0	50	0	100	31	40	0	0	0	31	25.2%
9	0	50	0	100	31	40	0	0	0	31	25.2%
10	0	40	0	100	12	40	0	32	0	50	27.4%

To provide a comprehensive overview of the binary classification results, we thoughtfully recorded values for True Positives (TP), True Negatives (TN), False Positives (FP), and False Negatives (FN). Furthermore, for a more detailed assessment, we compiled Tables 4.6 and 4.7, which outline the recall, precision, and accuracy metrics for both binary and family classifications, respectively. These tables serve as valuable references for understanding the effectiveness and reliability of our classification methods.

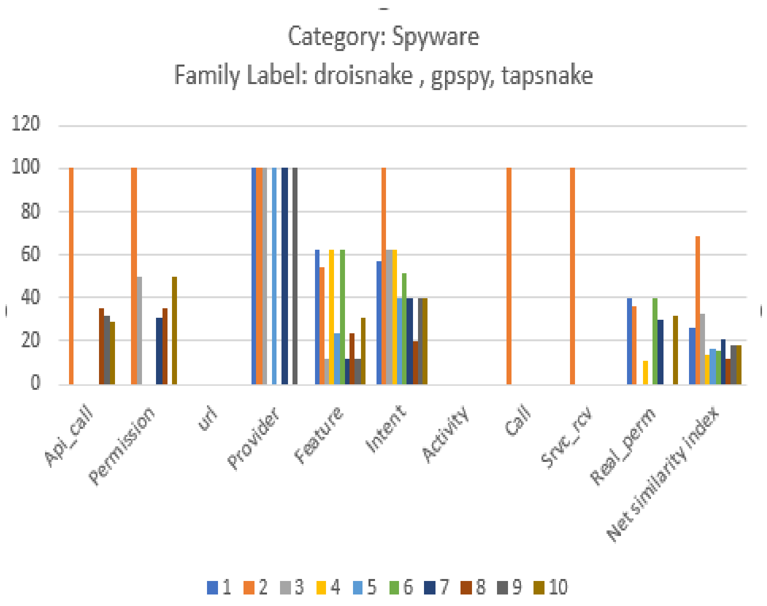

Figure 4.1: Graphical representation of Family Classification: Spyware

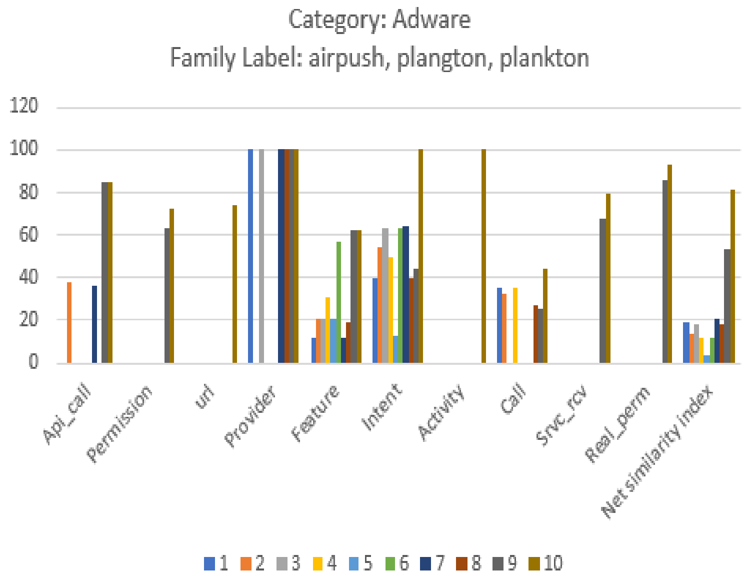

Figure 4.2: Graphical representation of Family Classification: Adware

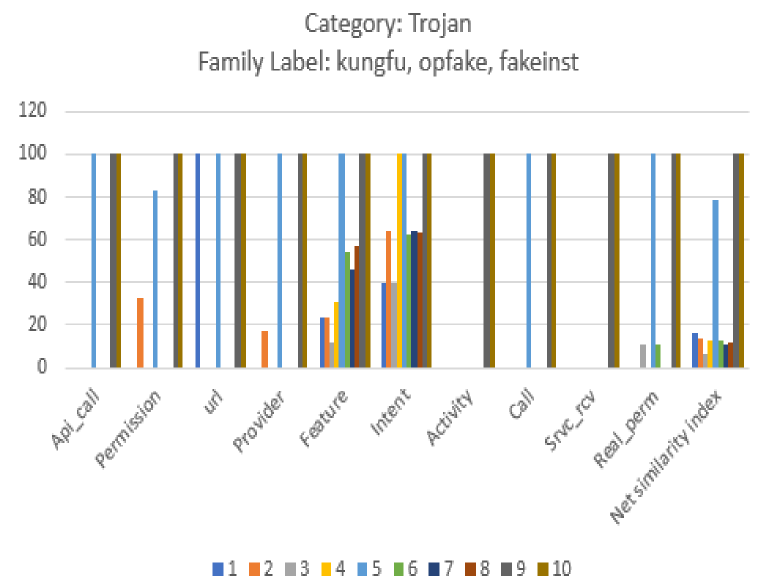

Figure 4.3: Graphical representation of Family Classification: Trojan

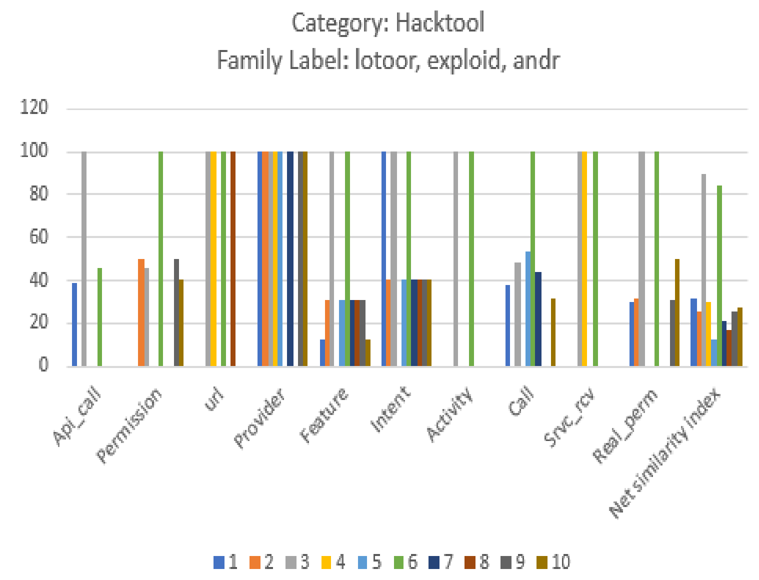

Figure 4.4: Graphical representation of Family Classification: Hacktool

The final segment of our research entails a comprehensive comparative analysis of existing studies. We have visualized this comparison through the use of graphical representations, specifically in the form of bar charts and line graphs, presented in Figure4.5 and Figure4.6. These visuals provide a clear and illustrative depiction of the data, facilitating a more in-depth understanding of the comparative findings.

Table 4.6: Results for Binary Classification of APK files

Method	**TP**	**FP**	**FN**	**TN**	**Recall**	**Precision**	**F1 Score**	**Accuracy**
With PFI	1.2%	3%	1.8%	94%	1.26%	28%	2.41%	95.2%
Without PFI	1.2%	2.4%	2.4%	95.18%	1.24%	33%	2.39%	96.4%

Table 4.7: Comparison Results for Family Classification and Variant Detection

Method	**TP**	**FP**	**FN**	**TN**	**Accuracy**	**Precision**	**Recall**	**F1 Score**
Karbab et al. [22] Base Paper I	-	-	-	-	-	89%	84%	86%
Shiel et al. [6] Base Paper II	40.25%	9.75%	9.75%	40.25%	80.5% & 94%	81%	81%	81%
Proposed Framework	13%	1.6%	1.6%	83%	96.67%	88%	88%	89.7%

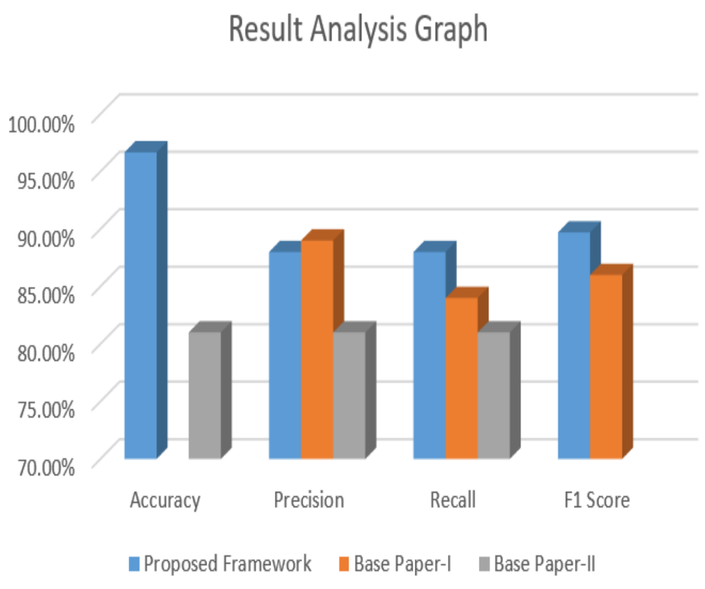

Figure 4.5: Bar Graph of Proposed Methodology with Existing Research

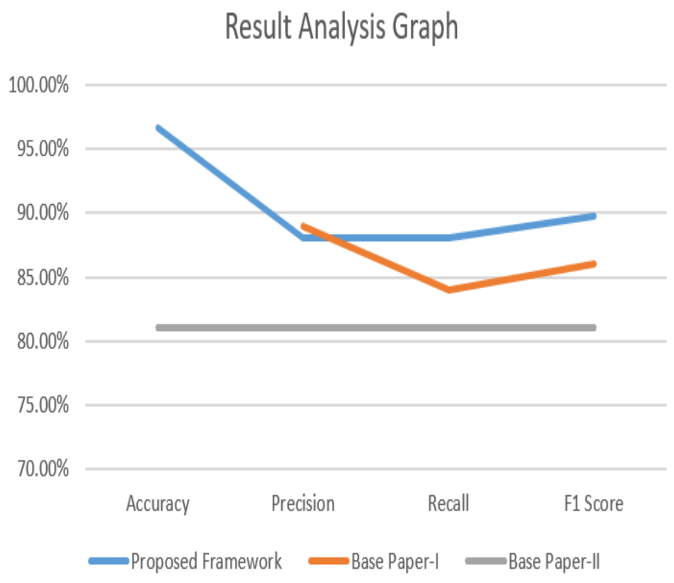

Figure 4.6: Line Graph of Proposed Methodology with Existing Research

CHAPTER 5

CONCLUSION & FUTURE WORK

The primary objective of this research was to enhance the detection of Android malware and its variants. Through the utilization of fuzzy hashes derived from the static features of APK files and their subsequent comparison with malware feature hashes, a substantial improvement in detection rates and accuracy was achieved. Similarity hash functions prove highly effective for malware detection without requiring the execution of files. While earlier approaches mainly focused on generating similarity hashes for entire executable files or specific file sections like files and directories, our approach is distinctive. We implemented a similarity hash algorithm on the static attributes of decompiled APK files. We then compared these hashes with those of known malicious APKs to calculate similarity scores. This approach enabled us to determine whether a test app is malicious and, if so, to classify it into the appropriate malware family. What makes our work remarkable is that it achieved a remarkable 96.67% accuracy improvement without relying on complex machine learning or deep learning techniques. The outcomes were encouraging, surpassing other state-of-the-art methods by a significant 2.67% margin.

Moreover, we conducted an in-depth analysis of APK file features to pinpoint which ones contribute effectively to detection and which ones introduce unnecessary noise. By reducing the use of inefficient features, we significantly enhanced the accuracy of classifying benign applications. In essence, our research not only presents a novel approach but also optimizes the detection process by focusing on the most relevant APK features. In today's rapidly evolving technological landscape, the sophistication of malware continues to advance. To ensure the continued success and relevance of our research, it becomes imperative to enrich malware repositories and databases with a broad spectrum of up-to-date malware hashes. This will enable our framework to adapt and thrive in the face of evolving threats.

The ever-evolving nature of malware necessitates a commitment to ongoing research and development. To further augment our research and enhance its practical applicability, future work should focus on enriching databases with up-to-date malware apk files and more analyzed features. The integration of a threat intelligence platform can also mark a significant step toward enhancing malware detection. By integrating such a platform, our system can dynamically extract and incorporate the latest information and intelligence related to emerging malware threats. This integration would play a pivotal role in continually populating our database with the most current and relevant malware hashes.

REFERENCES

[1] T. Baba, K. Baba, T. Yamauchi, Malware classification by deep learning using characteristics of hash functions, in: International Conference on Advanced Information Networking and Applications, Springer, 2022, pp. 480–491.

[2] Mobile operating system market share worldwide, Available at `https://gs.statcounter.com/os-market-share/mobile/worldwide` (December 6, 2022).

[3] Global android malware volume 2020, Available at `https://www.statista.com/statistics/680705/global-android-malware-volume/` (December 6, 2022).

[4] A. H. Galib, B. M. Hossain, A systematic review on hybrid analysis using machine learning for android malware detection, in: 2019 2nd International Conference on Innovation in Engineering and Technology (ICIET), IEEE, 2019, pp. 1–6.

[5] C. S. Yadav, S. Gupta, A review on malware analysis for iot and android system, SN Computer Science 4 (2) (2022) 118.

[6] I. Shiel, S. O'Shaughnessy, Improving file-level fuzzy hashes for malware variant classification, Digital Investigation 28 (2019) S88–S94.

[7] S. Lee, W. Jung, S. Kim, J. Lee, J.-S. Kim, Dexofuzzy: Android malware similarity clustering method using opcode sequence, Virus Bulletin (2019).

[8] P. Faruki, V. Laxmi, A. Bharmal, M. S. Gaur, V. Ganmoor, Androsimilar: Robust signature for detecting variants of android malware, Journal of Information Security and Applications 22 (2015) 66–80.

[9] L. Cai, Y. Li, Z. Xiong, Jowmdroid: Android malware detection based on feature weighting with joint optimization of weight-mapping and classifier parameters, Computers & Security 100 (2021) 102086.

[10] J. Zhang, Z. Qin, K. Zhang, H. Yin, J. Zou, Dalvik opcode graph based android malware variants detection using global topology features, IEEE Access 6 (2018) 51964–51974.

[11] A. Pektaş, T. Acarman, Learning to detect android malware via opcode sequences, Neurocomputing 396 (2020) 599–608.

[12] J. Tang, R. Li, Y. Jiang, X. Gu, Y. Li, Android malware obfuscation variants detection method based on multi-granularity opcode features, Future Generation Computer Systems 129 (2022) 141–151.

[13] J. Zhang, Z. Qin, H. Yin, L. Ou, K. Zhang, A feature-hybrid malware variants detection using cnn based opcode embedding and bpnn based api embedding, Computers & Security 84 (2019) 376–392.

[14] N. Kumari, M. Chen, Malware and piracy detection in android applications, in: 2022 IEEE 5th International Conference on Multimedia Information Processing and Retrieval (MIPR), IEEE, 2022, pp. 306–311.

[15] P. Yadav, N. Menon, V. Ravi, S. Vishvanathan, T. D. Pham, A two-stage deep learning framework for image-based android malware detection and variant classification, Computational Intelligence (2022).

[16] D. Vasan, M. Alazab, S. Wassan, H. Naeem, B. Safaei, Q. Zheng, Imcfn: Image-based malware classification using fine-tuned convolutional neural network architecture, Computer Networks 171 (2020) 107138.

[17] S. A. Roseline, S. Geetha, S. Kadry, Y. Nam, Intelligent vision-based malware detection and classification using deep random forest paradigm, IEEE Access 8 (2020) 206303–206324.

[18] M. Xiao, C. Guo, G. Shen, Y. Cui, C. Jiang, Image-based malware classification using section distribution information, Computers & Security 110 (2021) 102420.

[19] A. Darem, J. Abawajy, A. Makkar, A. Alhashmi, S. Alanazi, Visualization and deep-learning-based malware variant detection using opcode-level features, Future Generation Computer Systems 125 (2021) 314–323.

[20] L. Nataraj, T. M. Mohammed, T. Nanjundaswamy, S. Chikkagoudar, S. Chandrasekaran, B. Manjunath, Omd: Orthogonal malware detection using audio, image, and static features, in: MILCOM 2021-2021 IEEE Military Communications Conference (MILCOM), IEEE, 2021, pp. 703–708.

[21] Y. Zhang, W. Ren, T. Zhu, Y. Ren, Saas: A situational awareness and analysis system for massive android malware detection, Future Generation Computer Systems 95 (2019) 548–559.

[22] E. B. Karbab, M. Debbabi, D. Mouheb, Fingerprinting android packaging: Generating dnas for malware detection, Digital Investigation 18 (2016) S33–S45.

[23] B. Jin, J. Choi, H. Kim, J. B. Hong, Fumvar: a practical framework for generating fully-working and u nseen m alware var iants, in: Proceedings of the 36th Annual ACM Symposium on Applied Computing, 2021, pp. 1656–1663.

[24] M. İbrahim, B. Issa, M. B. Jasser, A method for automatic android malware detection based on static analysis and deep learning, IEEE Access 10 (2022) 117334–117352.

[25] T. Baba, K. Baba, T. Yamauchi, Malware classification by deep learning using characteristics of hash functions, in: International Conference on Advanced Information Networking and Applications, Springer, 2022, pp. 480–491.

[26] S. Choi, Combined knn classification and hierarchical similarity hash for fast malware detection, Applied Sciences 10 (15) (2020) 5173.

[27] J. Choi, D. Shin, H. Kim, J. Seotis, J. B. Hong, Amvg: Adaptive malware variant generation framework using machine learning, in: 2019 IEEE 24th Pacific Rim International Symposium on Dependable Computing (PRDC), IEEE, 2019, pp. 246–24609.

[28] A. P. Namanya, I. U. Awan, J. P. Disso, M. Younas, Similarity hash based scoring of portable executable files for efficient malware detection in iot, Future Generation Computer Systems 110 (2020) 824–832.

[29] N. Naik, P. Jenkins, N. Savage, L. Yang, T. Boongoen, N. Iam-On, Fuzzy-import hashing: A malware analysis approach, in: 2020 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE), IEEE, 2020, pp. 1–8.

[30] N. Naik, P. Jenkins, N. Savage, L. Yang, K. Naik, J. Song, T. Boongoen, N. Iam-On, Fuzzy hashing aided enhanced yara rules for malware triaging, in: 2020 IEEE Symposium Series on Computational Intelligence (SSCI), IEEE, 2020, pp. 1138–1145.

[31] A. Altaher, An improved android malware detection scheme based on an evolving hybrid neuro-fuzzy classifier (ehnfc) and permission-based features, Neural Computing and Applications 28 (2017) 4147–4157.

[32] S. Gupta, H. Sharma, S. Kaur, Malware characterization using windows api call sequences, in: Security, Privacy, and Applied Cryptography Engineering: 6th International Conference, SPACE 2016, Hyderabad, India, December 14-18, 2016, Proceedings 6, Springer, 2016, pp. 271–280.

[33] Y. Li, J. Jang, X. Ou, Topology-aware hashing for effective control flow graph similarity analysis, in: Security and Privacy in Communication Networks: 15th EAI International Conference, SecureComm 2019, Orlando, FL, USA, October 23-25, 2019, Proceedings, Part I 15, Springer, 2019, pp. 278–298.

[34] E. B. Karbab, M. Debbabi, A. Derhab, D. Mouheb, E. Billah Karbab, M. Debbabi, A. Derhab, D. Mouheb, Robust android malicious community fingerprinting, Android Malware Detection using Machine Learning: Data-Driven Fingerprinting and Threat Intelligence (2021) 61–99.

[35] N. Naik, P. Jenkins, N. Savage, L. Yang, K. Naik, J. Song, Augmented yara rules fused with fuzzy hashing in ransomware triaging, in: 2019 IEEE Symposium Series on Computational Intelligence (SSCI), IEEE, 2019, pp. 625–632.

[36] H. Ali, K. Batool, M. Yousaf, M. Islam Satti, S. Naseer, S. Zahid, A. A. Gardezi, M. Shafiq, J.-G. Choi, Security hardened and privacy preserved android malware detection using fuzzy hash of reverse engineered source code., Security & Communication Networks (2022).

[37] N. Sarantinos, C. Benzaïd, O. Arabiat, A. Al-Nemrat, Forensic malware analysis: The value of fuzzy hashing algorithms in identifying similarities, in: 2016 IEEE Trustcom/BigDataSE/ISPA, IEEE, 2016, pp. 1782–1787.

[38] S. Hiruta, Y. Yamaguchi, H. Shimada, H. Takakura, Evaluation on malware classification by combining traffic analysis and fuzzy hashing of malware binary, in: Proceedings of the International Conference on Security

and Management (SAM), The Steering Committee of The World Congress in Computer Science, Computer . . . , 2015, p. 89.

[39] H. Rafiq, N. Aslam, M. Aleem, B. Issac, R. H. Randhawa, Andromalpack: enhancing the ml-based malware classification by detection and removal of repacked apps for android systems, Scientific Reports 12 (1) (2022) 19534.

[40] O. Mirzaei, G. Suarez-Tangil, J. M. de Fuentes, J. Tapiador, G. Stringhini, Andrensemble: Leveraging api ensembles to characterize android malware families, in: Proceedings of the 2019 ACM Asia conference on computer and communications security, 2019, pp. 307–314.

[41] M. Kida, O. Olukoya, Nation-state threat actor attribution using fuzzy hashing, IEEE Access 11 (2022) 1148–1165.

[42] J. Oliver, M. Ali, J. Hagen, Hac-t and fast search for similarity in security, in: 2020 International Conference on Omni-layer Intelligent Systems (COINS), IEEE, 2020, pp. 1–7.

[43] P. Wu, D. Liu, J. Wang, B. Yuan, W. Kuang, Detection of fake iot app based on multidimensional similarity, IEEE Internet of Things Journal 7 (8) (2020) 7021–7031.

[44] R. Kumar, K. Sethi, N. Prajapati, R. R. Rout, P. Bera, Machine learning based malware detection in cloud environment using clustering approach, in: 2020 11th International Conference on Computing, Communication and Networking Technologies (ICCCNT), IEEE, 2020, pp. 1–7.

[45] J. Akram, Z. Shi, M. Mumtaz, P. Luo, Droidsd: An efficient indexed based android applications similarity detection tool., Journal of Information Science & Engineering 36 (1) (2020).

[46] J. Akram, M. Mumtaz, G. Jabeen, P. Luo, Droidmd: an efficient and scalable android malware detection approach at source code level, International Journal of Information and Computer Security 15 (2-3) (2021) 299–321.

[47] M. Botacin, V. H. G. Moia, F. Ceschin, M. A. A. Henriques, A. Grégio, Understanding uses and misuses of similarity hashing functions for malware detection and family clustering in actual scenarios, Forensic Science International: Digital Investigation 38 (2021) 301220.

[48] N. Naik, P. Jenkins, N. Savage, A ransomware detection method using fuzzy hashing for mitigating the risk of occlusion of information systems, in: 2019 international symposium on systems engineering (ISSE), IEEE, 2019, pp. 1–6.

[49] M. İbrahim, B. Issa, M. B. Jasser, A method for automatic android malware detection based on static analysis and deep learning, IEEE Access 10 (2022) 117334–117352.

[50] J. Upchurch, X. Zhou, Variant: a malware similarity testing framework, in: 2015 10th International Conference on Malicious and Unwanted Software (MALWARE), IEEE, 2015, pp. 31–39.

APPENDIX A

```
import ssdeep
import json

# Define the list of dictionaries to compare against
with open('fuzzy_HASHES.json', 'r') as file1:
    data1 = json.load(file1)

# Define the dictionary to compare against the list
input_dict = {
        "sha256": "4bf0f0cb2b10c9bad568eec092e600
        f51bff60670103e91431e3bf6e92d4777b",
        "api_call": "3:KyRMjKTA5rlRVmJ9jiBXXBwK17vPg/
                                        9B5TRh6ibBA
        plrNCi9BIAsmEsORM:EKMplRMJ9j0XBwCvY/9B/AKmzrsgBIjc",
        "permission": "3:icAIDkRgR6hFcaMBIDkRgR6T/
                                        cRcaMBIDs6bgxFi9BIDy31X
        BID3jgSyq3Tcxkn:icAxRgIWaMBxRgO/cyaMB5a0Fi9BnZBQ",
        "url": "3:5OGKs0F30F30F30F30n:TKlMMMk",
        "provider": "3::",
        "feature": "3:icAnDRKQahAKgBnDJKTRMXHLhXBnDJ
        KTRMR1HBnDJKTRMvSM
        O:icANK0pBVKTWL9BVKTyHBVKTJMO",
        "intent": "3:icAGRAlwGxvR3Tcxk6BGRAlwGxgkpKMBGRAlyQKGn
        :icAGewGxvR3QRBGewGxhpKMBGeyQKGn",
        "activity": "3:1ORZcn:0cn",
        "call": "3:nzjN2nE:n8E",
        "service_receiver": "3:5LORTn:5LORT",
        "real_permission": "3:icAIDkRgR6hFcaMBIDy31XBIDs6bgxn
        :icAxRgIWaMBnZB5a0n"
    }

# Define keys to skip during comparison
```

```
keys_to_skip = ["sha256", "submission_date", "label"]

# Function to compare dictionaries using ssdeep
def compare_dicts(dict1, dict2):
    match_scores = {}
    sha256_value = dict2["sha256"]
    for key in dict1:
        if key not in keys_to_skip:
            match_score = ssdeep.compare(dict1[key], dict2[key]
                                             )
            match_scores[key] = match_score
    return sha256_value, match_scores

output_file = "result6.txt"
with open(output_file, "w") as f:
# Compare the input_dict with each dictionary in the
                                list_of_dicts
    for  other_dict in data1:
        sha256_value, match_scores = compare_dicts(input_dict,
                                        other_dict)
        print(f"Comparison with sha256 value '{sha256_value}':"
                                        , file=f)
        for key, score in match_scores.items():
            print(f"{key}: {score}", file=f)
        print("", file=f)
```

```
# Data sample for malware category and family classes
    [{"sha256": "a2b4f6918034895
    fd3948123ec18bb46fcd1d91468f
    5c7d829e3b220e680d280",
    "malware_category": ["trojan","adware"],
    "malware_family": ["airpush", "plangton", "plankton"]},
    {"sha256": "5d7ba82561d1c161c16f9d5a719894
    b8784c8d1af3faac3f127410751e7cc80e",
    "malware_category": ["trojan","adware"],
    "malware_family": ["airpush", "plangton", "plankton"]},
    {"sha256": "ac3561b823fc0aaa9c3b15fb8e1e710e79899479653a
    71b74644ab9aa201e197", "malware_category": ["trojan","
                                    adware"],
    "malware_family": ["airpush", "plangton", "plankton"]},
    {"sha256": "763e46727b29a0fd994b9fbb1ea9
    346b03c31a54d19ffcde66e4d15498e49a11",
    "malware_category": ["trojan","adware"],
    "malware_family": ["airpush", "plangton", "plankton"]},
    {"sha256": "2c745f3ae79d2e078e7e38c21cb12a
    ecd7be09a30fc7f9ad02dd5344e1015b6e",
    "malware_category": ["trojan","adware"],
    "malware_family": ["airpush", "plangton", "plankton"]},
    {"sha256": "666b9646e8bf8b3610dc72ae10e38a2c5c
    16c836336960ab64ab6c748f788126",
    "malware_category": ["trojan","adware"],
    "malware_family": ["airpush", "plangton", "plankton"]},
    {"sha256": "85eb6c81ccc7fc19e20c55e0aea5a
    5afbe3d45d8ac502de1ea72376303965aea",
    "malware_category": ["trojan","adware"],
    "malware_family": ["airpush", "plangton", "plankton"]},
    {"sha256": "d521b8787d8641993c14f4feb103
    d1b4114eb6bce5184f04409c800ba898f9d9",
    "malware_category": ["trojan", "virus", "adware"],
    "malware_family": ["ginmaster", "gingermaster", "gmaster"]}
                                    ,
    {"sha256": "e321f789d60e11f39f8d73a4f2c2dd45
    d90975d8bc6bd7504d06d8aa6a270e84", "malware_category": "
                                    trojan ",
    "malware_family": ["plankton", "plangton", "andr"]}
]
```