Smali opcode based Android Malware detection and Obfuscation Identification

Abhishek Anand (abhisheka.ph21.cs@nitp.ac.in)
National Institute of Technology Patna
Jyoti Prakash Singh
National Institute of Technology Patna

Research Article

Keywords: Android Malware, Smali,Opcodes, Obfuscation, Malware Families

Posted Date: October 30th, 2023

DOI: https://doi.org/10.21203/rs.3.rs-3493657/v1

Additional Declarations: No competing interests reported.

Smali opcode based Android Malware detection and Obfuscation Identification

Abhishek Anand[1*] and Jyoti Prakash Singh[1††]

[1]Computer Science and Engineering, National Institute of Technology Patna, Patna, 800005, Bihar, India.

*Corresponding author(s). E-mail(s): abhisheka.ph21.cs@nitp.ac.in;
Contributing authors: jps@nitp.ac.in;
†These authors contributed equally to this work.

Abstract

The Android platform's open-source nature makes it a prime target for attackers seeking to exploit vulnerabilities. The practice of reverse engineering in Android applications further increases this vulnerability, creating a lucrative ground for exploitation and attack. Malware developers use various obfuscation techniques to protect applications from reverse engineering attempts. These same obfuscation techniques are utilized by malware creators to hide malicious code within the application's structure. Obfuscation introduces useless code and concealed features during feature extraction, making it difficult for conventional malware analysis methods to recognise the application and resulting in a high rate of false negatives. To address this, this paper introduces an innovative Smali opcode-based model, specifically designed to address the complexity of obfuscation techniques during both binary and familial classification. The core objective is to design a lightweight model capable of classifying malware and benign applications, alongside robust familial classification. Moreover, the model is also equipped to identify the specific obfuscation technique employed in a given malware application. We have meticulously implemented and rigorously evaluated the proposed model using two distinct datasets encompassing obfuscated and non-obfuscated samples. The experimental findings affirm the model's performance, surpassing existing state-of-the-art Android malware classifiers. Notably, the model achieves an impressive binary classification accuracy of 99.4%.

Keywords: Android Malware, Smali,Opcodes, Obfuscation, Malware Families

1 Introduction

The proliferation of malware attacks on mobile platforms, particularly on Android, is a growing concern due to the platform's popularity and extensive availability of applications on various sources, including the official Google Play store and third-party markets. Android has dominated the mobile platform market, accounting for 74.13% of the market share in terms of mobile operating systems worldwide as of December 2019, according to Statista [1]. The widespread adoption of Android also attracts malware adversaries who exploit reverse engineering as a relatively simple method to develop Android malware [2]. To rebuttal traditional malware detection methodologies, such as signature-based and machine learning or deep learning-based systems, Android malware authors have begun employing various obfuscation techniques. Obfuscation techniques are utilized to conceal malicious scripts within the application code, making it more challenging for traditional analysis methods to accurately detect and analyze malware behaviour [3, 4]. The obfuscation process makes the malware code complex and generates futile and camouflaged features, which hinder the effectiveness of malware detection techniques. This obfuscation-driven complexity creates difficulties for accurate behavioural analysis of Android applications, complicating the task of identifying malicious behaviour effectively.

The traditional malware detection and analysis methods [5–7] couldn't handle obfuscated malwares properly. Traditional techniques can be broadly classified into two categories: string-based approaches and graph-based techniques. String-based approaches focus on extracting features such as permissions [6], intents and API calls [5, 7] from the application's code. However, these methods lack contextual and structural information, which makes them susceptible to being circumvented by obfuscation techniques. Obfuscation can modify or hide the original features, making it difficult for string-based approaches to accurately analyze obfuscated malware. Graph-based techniques, on the other hand, represent the application's program structure as a graph and use graph matching for classification [8, 9]. This method can outperform string-based approaches since it considers the program's schematics. However, after obfuscation, the structure of the graph can change significantly, and the extracted features may not be the same as before obfuscation. This can result in the inclusion of bogus and irrelevant features, making it challenging for malware detection techniques to perform accurate behavioural analysis.

Obfuscation in Android applications can be viewed as a double-edged sword, serving both legitimate Android developers and malicious actors, such as malware developers. Obfuscation techniques aim to protect the original code from being easily reverse-engineered by unauthorized parties, ensuring the security of intellectual property and preventing code theft. At the same time, these techniques can also be exploited by malware developers to evade detection and make their malicious code more challenging to analyze [3]. Obfuscation in Android applications encompasses various techniques, some of which are relatively straightforward and others more complex. These obfuscation techniques serve to protect the integrity and intellectual property of Android applications, particularly from reverse engineering attempts. However, they can also be employed by malware developers to make their code more elusive and evade detection by security tools.

Also understanding malware families is a crucial aspect of Android malware analysis. By identifying and categorizing malware into distinct families, developers and security experts can gain valuable insights into the behaviour, characteristics, and propagation methods of different types of malware. This knowledge is essential for developing more effective and targeted detection and mitigation strategies. However, the inconsistency in naming malware families by different antivirus companies presents a major challenge. The lack of standardized naming conventions can lead to confusion and hamper the collaborative efforts of the cybersecurity community. It can also make it difficult for researchers and developers to effectively share information, tools, and insights related to specific malware families. In light of the mentioned challenges, the paper proposes a comprehensive approach to handle obfuscated malwares and addresses familial classification. The main contributions of the research are summarized as follows:

1. Designing malware analysis model leveraging Smali opcodes for classifying malware and benign applications and also incorporating family classification.
2. Recognizing the existence of obfuscation within the malware dataset.
3. Identifying the specific obfuscation techniques (i.e. Code obfuscation, encryption, renaming, reflection and trivial) employed by malware applications.
4. Introducing a lightweight malware analysis model that requires minimal preprocessing and imposes negligible training and testing overheads.

The proposed model is evaluated on two distinct datasets: one containing non-obfuscated samples (Drebin [5]) and the other consisting of obfuscated samples (AndroOBFS [10]). Additionally, separate benign datasets (CCCS-CIC-AndMal-2020 [11, 12], CICMalDroid2020 [13]) and Androzoo [14] is used for binary classification of malware and benign applications.

The following are the remaining sections: Section 2 discusses the related work on malware analysis and familial classification. Section 3 provides a comprehensive and detailed explanation of the proposed model, Section 4 presents the result of the proposed model, Section 5 goes over an extensive analysis and discussion on the outcomes of the paper and Section 6 concludes the paper.

2 Related Work

Static analysis is a method used to detect and analyze malware based on the information present in an APK (Android Package) file without actually executing the application. This approach is effective for identifying potentially harmful code and malicious behaviour before the application is installed on a device. This analysis technique is widely explored in this section.

Kang et al. [15] introduced a novel approach for Android malware detection and categorization based on n-gram Dalvik opcode analysis. The researchers utilized n-gram frequencies of opcode occurrences as features for their machine-learning classifiers. The machine learning classifiers employed in the study included Naive Bayes, Support Vector Machine (SVM), and Random Forest, among others. To evaluate the effectiveness of their approach, they conducted experiments on a dataset consisting of 2,520 data samples and achieved an F-measure of 98%. Bakshshinejad et al. [16] introduced an innovative heuristic malware identification technique centered around compression methods. The researchers leveraged Smali opcodes to compress models, effectively optimizing the accuracy of the model and achieving a remarkable maximum accuracy of 95.8%. McLaughlin et al. [17] introduced an innovative Android malware detection system based on deep convolutional neural networks (CNNs). Their approach involved using training pipelines to learn relevant features and subsequently perform the classification of Android applications with an accuracy of 69%. Zegzhda et al. [18] employed a Convolutional Neural Network (CNN) model that utilized RGB images generated from API sequences. These API sequences were extracted from Smali code, which is a representation of the Dalvik bytecode used in Android applications. The process of generating RGB images from API sequences allowed the researchers to transform the raw code data into a format suitable for CNNs and achieve an accuracy of 93.64%. Chen et al. [19] proposed a lightweight model for Android malware detection, which centered around using opcode sequences extracted from Android Dalvik executable files. The researchers employed the concept of n-grams to extract features from these opcode sequences, enabling the model to understand patterns and behaviours in Android applications effectively. To evaluate the performance of their model, they employed a range of classifiers, including K-Nearest Neighbors (KNN), Naive Bayes, and Random Forest, among others and gained the best accuracy of 99.4%. Sihag et al. [20] proposed a Smali opcode-based model for malware family classification. They leveraged these opcodes to build a model capable of classifying Android malware into different families. To calculate similarity among n-gram-based attributes, they utilized bloom filters, which are space-efficient probabilistic data structures. They found the best accuracy of 94.34%. Niu et al. [21] investigated Android malware detection using Function Call Graph (FCG) generated through static analysis of opcodes. They explored the behavioural features of malware applications by analyzing the FCG, which provides insights into the interactions between functions and methods in an application's code. They gained the best accuracy of 97% in classifying Android malware. Iadarola et al. [22] presented a novel approach to malware family classification by representing Android applications as images. The researchers converted the applications into image format and utilized them as input to deep learning models for prediction. They also exploited the activation maps generated by the deep learning model to identify and highlight potential malicious classes within the application images and achieve an accuracy of 94.4%.

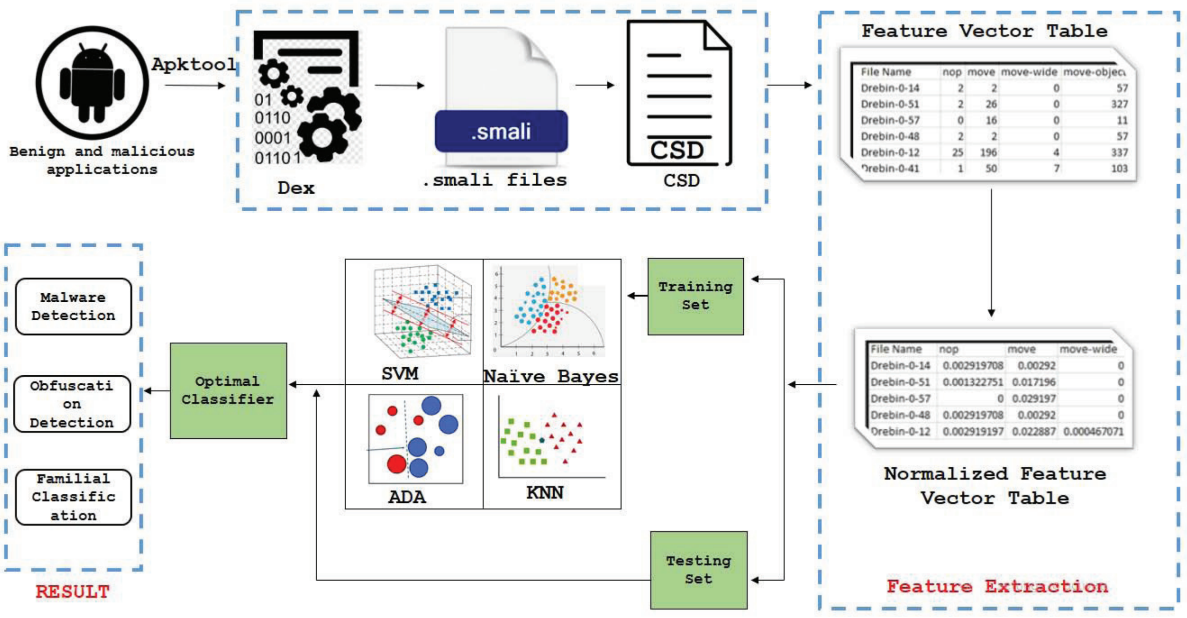
Sihag et al. [23] employed opcode segments for feature characterization in the context of Android malware detection. They performed semantics-based simplification of Smali opcodes, which involves reducing the complexity of opcode sequences while preserving their essential meaning and behaviour. To assess the effectiveness of their approach, they evaluated their models on different obfuscation techniques commonly employed by malware authors, such as encryption, reflection, and trivial obfuscation and scored the best accuracy of 98.6%. Nivedha et al. [24] collected a diverse set of static features from Android applications, including permissions, Intent usage, API calls, and n-gram opcodes. To effectively classify Android applications based on these static features, they employed a deep learning model and got an accuracy of 99.37%. Millar et al. [25] proposed a deep learning-based method for classifying Android malware and benignware in zero-day scenarios. They utilized Convolutional Neural Networks (CNNs) to analyze three critical features: permissions, opcodes, and API sequence calls, extracted from Android applications. They evaluated the performance of their approach on two datasets: AMD [26] and Drebin [5]. On the AMD dataset, the method achieved a weighted average detection rate of 81%. On the Drebin dataset, which is widely used for Android malware research, the method achieved an even higher weighted average detection rate of 91%. Lajevardi et al. [27] introduced a dynamic and behavior-based malware detection approach named Markhor. This approach leverages system call data dependency and system call control dependency sequences to construct a weighted list of patterns associated with malicious behaviors. The model used a fuzzy algorithm, a mathematical technique used to handle uncertainty, approximate reasoning and achieved an accuracy of 98.20%. Yang et al. [28] introduced an innovative approach known as the adversarial instruction technique, which leverages machine learning and deep learning methods for Android malware analysis. The researchers applied this technique to three common types of malware samples: Trojan horses, Ransomware, and Backdoors, while ensuring the original attack functionality of these malware categories was maintained. The model achieved a mean F1-score of 93.94%.

Khalid et al. [29] introduced an opcode sequence-based malware detection system specifically designed to identify obfuscated malware samples. Obfuscation is a technique used by malware authors to disguise their code and evade traditional detection methods. They focused on studying the impact of various code obfuscation schemes on sequential opcode-based analytic systems. The proposed model achieved an impressive accuracy of 97.2% when classifying samples into two class categories: malware and benign applications. Hashemi et al. [30] introduced a unique approach for malware detection using image fusion techniques. The researchers converted executable files into RGB images, effectively transforming code into a visual format. They then utilized a deep convolutional neural network (CNN) to analyze and detect malware based on these image representations and reached an accuracy of 97.30%. Chysi et al. [31] developed a novel approach for analyzing Android applications by generating system-call dependency graphs and constructing flow maps that represent maximum flow within these graphs. The system-call dependency graphs capture the relationships between different system calls used by an application. They employed a machine learning technique and conducted a series of five-fold cross-validation experiments and gained the maximum accuracy of 94.80%. Nguyen et al. [32] focused on utilizing Generative Adversarial Networks (GANs) for multiclass classification and evaluated their effectiveness in comparison to popular machine learning techniques. Their work particularly examined the utility of GAN models for adversarial attacks on image-based malware detection. The model achieves accuracy of more than 99% in familial classification. Maholtra et al. [33] employed a functional call graph, which represents a set of program functions and their interprocedural calls, as a basis for their analysis. They utilized graph neural networks to train and classify Android applications based on their functional call graphs to achieve an accuracy of 97.69%. Qiu et al. [34] introduced a novel Multiview Feature Intelligence (MFI) framework for Android malware detection. The MFI framework involved reverse engineering the APK files to extract multiple types of features, including semantic string features, opcode sequential features, and API call graph features. To classify Android applications based on these extracted features, they employed deep neural networks as a classifier and achieved an F1 score of 0.94. Kang et al. [15] proposed an n-opcode analysis-based model for classifying Android applications into malware and benignware categories. The researchers leveraged machine learning classifiers to analyze and categorize the applications based on opcode features. They utilized 10-gram opcode features and achieved an F1-score of 98%. Manzil et al. [35] introduced an innovative approach involving Huffman encoding to generate feature vectors for differentiating Android malware categories. They focused on classifying specific types of malware, such as riskware, adware, SMS malware, and banking malware. To create feature vectors, they utilized system call frequencies and then evaluated machine learning and deep learning methods to classify the various malware categories to get a maximum accuracy of 98.70%. Huang et al. [36] focused on two main areas: clone detection and obfuscation analysis. They developed a clone detection framework that utilized three deep learning-based clone detectors and compared their performance with traditional clone detection methods. Clone detection involves identifying duplicate or similar sections of code within a codebase, which is important for identifying code redundancy and maintaining code quality. They delved into the impact of code obfuscation on clone detection.

Several research studies have been presented in the literature, focusing on malware detection and familial classification. However, most of these works rely on numerous features for static malware analysis, often coupled with complex preprocessing steps and resource-intensive deep-learning models. This increases the preprocessing and execution time overheads for the model. The proposed work centres around the development of a lightweight model that efficiently addresses challenges encompassing malware classification, familial classification, and obfuscation identification. This approach aims to strike a balance between accuracy and computational efficiency, contributing to a more streamlined solution.

Algorithm 1 Reverse Engineering on Android Application(.apk file)

```
Data: APK file of Mal, Ben
Result: Classes.dex file and .smali files
Initialization: X ← 1, $\sum_{i=1}^{5560}(Mal_i), \sum_{i=1}^{8031}(Ben_i)$;
$Total_{APK} = \sum_{i=1}^{5560}(Mal_i) + \sum_{i=1}^{8031}(Ben_i)$;
1: for every X ∈ $Total_{APK}$ do
2:     $Output_{folder} \leftarrow ApkTool(X)$          ▷ This will give Dex.class
3:     Change extension of X from .apk to .zip;
4:     $Temp_{folder} \leftarrow zip(X)$;
5:     Classes.dex from $Temp_{folder}$ to $Output_{folder}$;
6:     X ← X+1;
7: end for
```



File Name	nop	move	move-wide	move-objec
Drebin-0-14	2	2	0	57
Drebin-0-51	2	26	0	327
Drebin-0-57	0	16	0	11
Drebin-0-48	2	2	0	57
Drebin-0-12	25	196	4	337
Drebin-0-41	1	50	7	103

File Name	nop	move	move-wide
Drebin-0-14	0.002919708	0.00292	0
Drebin-0-51	0.001322751	0.017196	0
Drebin-0-57	0	0.029197	0
Drebin-0-48	0.002919708	0.00292	0
Drebin-0-12	0.002919197	0.022887	0.000467071

Fig. 1 Proposed Architecture

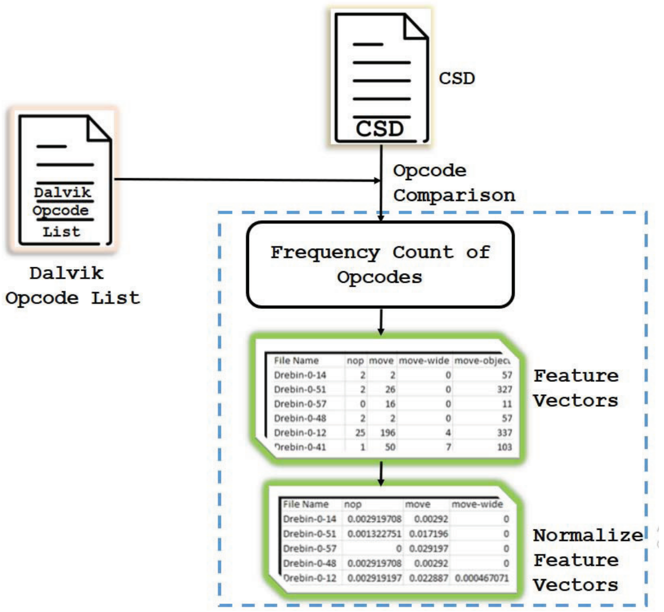
3 Proposed Methodology

3.1 Overview

The proposed methodology is centered around the utilization of Smali opcodes to classify traditional malware and benign applications. The model is not only capable of distinguishing between these two categories but also effectively addresses the challenges posed by obfuscated malware. It differentiates between obfuscated and non-obfuscated malware applications and goes further by identifying the specific obfuscation techniques employed on the Android applications. The proposed model excels in familial classification, a task that involves categorizing malware into specific families based on their shared characteristics and behaviours. In essence, the methodology encompasses three primary tasks: malware detection prediction, familial classification, and obfuscation detection. This complete architecture is visually represented in Figure 1, showcasing the flow and interconnections of these tasks within the proposed model. As depicted in the architectural diagram, the process begins with the extraction of the dex bytecode file from a given APK sample. This bytecode is then utilized to extract .smali files, which are instrumental in creating a Concatenated Smali Document (CSD). Each CSD encapsulates the combined content of all Smali files within the APK. Subsequently, Opcodes are extracted from the CSDs by comparing them with the Dalvik opcode list [37] and, the frequency of each opcode's occurrence is tallied

Algorithm 2 : CSD Generation

```
Input: sample.smali
Output: Conacat Smali Document (CSD) of the sample
Initialize CSD file
1: for each application directory do
2:     for files in files do
3:         if $file$ endswith .smali then
4:             append Smali files to CSD
5:         else
6:             Exit
7:         end if
8:     end for
9: end for
```



File Name	nop	move	move-wide	move-objec
Drebin-0-14	2	2	0	57
Drebin-0-51	2	26	0	327
Drebin-0-57	0	16	0	11
Drebin-0-48	2	2	0	57
Drebin-0-12	25	196	4	337
Drebin-0-41	1	50	7	103

File Name	nop	move	move-wide
Drebin-0-14	0.002919708	0.00292	0
Drebin-0-51	0.001322751	0.017196	0
Drebin-0-57	0	0.029197	0
Drebin-0-48	0.002919708	0.00292	0
Drebin-0-12	0.002919197	0.022887	0.000467071

Fig. 2 Flow Chart of Normalized Feature Vector generation from CSD

to generate a feature vector table. In this table, each individual opcode functions as a distinct feature. These features serve as the input for various machine learning classifiers employed in the model. Obfuscation techniques pose significant challenges in the field of malware analysis. These techniques can obscure and complicate the analysis process, making it harder to detect malicious behaviour. However, the proposed model significantly mitigates these threats.

3.2 Feature Extraction

In the proposed methodology, we have established a Smali opcode feature-based system for detecting and classifying malicious and benign applications. The process initiates with the decompilation of the target APK file through reverse engineering, which is executed using the Apktool [38] within the Kali Linux environment using Algorithm 1. Once the decompilation is accomplished, the next step involves the extraction of all Smali files residing within various directories of the decompiled APK.

All extracted Smali files are aggregated to create a single Concatenated Smali Document (CSD). Generation of CSD from different .smali files in various folders and subfolders of decompiled APK is done following Algorithm 2. Subsequently, leveraging the Dalvik opcode list, we systematically extract all opcodes present within the CSD. Each opcode from the Dalvik opcode list is compared with CSD and to quantify the significance of each opcode, we perform a frequency analysis, counting the occurrences of each opcode. This process results in the generation of a comprehensive feature table that encapsulates the frequency distribution of opcodes extracted from the CSD. To negate the variable size of APK files, the feature table is normalized using equation 1.

$$X' = \frac{X}{\sum_{i=1}^{n}(X_n)} \tag{1}$$

The feature extraction process is demonstrated in the flowchart shown in Figure 2. By structuring the methodology around the utilization of Smali opcodes and their corresponding frequency distributions, the model gains valuable insights into the behaviour and characteristics of the analyzed applications.

3.3 Classification Model

In our implementation, we integrate machine learning techniques to comprehensively analyze and classify both traditional and obfuscated malwares. Additionally, the approach extends to the familial classification of malware instances. The methodology involves the selection and utilization of eight distinct classifiers: Logistic Regression (LR), K-Nearest Neighbor (KNN), Naive Bayes (NB), Support Vector Machine (SVM), Random Forest (RF), Decision Tree (DT), and AdaBoost (AB). These classifiers are employed in the context of supervised learning, where they learn patterns from labelled data during the training phase and apply their learned knowledge to evaluate and classify unseen testing data. Throughout the process, sensitive behavioural features extracted from Android applications play a crucial role. These features enable the machine learning classifiers to effectively differentiate between various application types and categories.

4 Experimental Evaluation and Results

In this section of the paper, we begin by introducing the datasets and matrices that are used in experimental analysis. Subsequently, we delve into the evaluation of our proposed methodology, addressing the following research questions:

- RQ1: What is the performance of the proposed model on classifying malware and benign applications on obfuscated and non-obfuscated malwares?
- RQ2: What is the performance of the proposed model on classifying obfuscated and non-obfuscated malwares?
- RQ3: How well proposed model effectively distinguish different obfuscation techniques employed on malicious samples?
- RQ4: How well proposed model identify the malware families and classify them?
- RQ5: What is the runtime overhead of the proposed model for classifying Android malware?

4.1 Dataset and evaluation metrics

To evaluate the proposed model a widely used verified malware dataset Drebin [5] and AndroOBFS [10] is used. The utilization of the Drebin and AndroOBFS datasets for evaluating the proposed model provides a comprehensive assessment of its performance across different scenarios. These datasets offer a range of challenges and complexities commonly encountered in Android malware analysis. The Drebin dataset is a widely recognized and verified malware dataset, containing 5560 malware samples. These samples span across 179 distinct malware families. The malware samples within the Drebin dataset were released in the year 2014. This dataset serves as a benchmark for evaluating the effectiveness of malware detection models. The AndroOBFS dataset, released in the year 2020, consists of 15479 obfuscated malware samples. These samples are generated through the application of various obfuscation techniques on samples from Androzoo and VirusShare.com. The dataset encompasses multiple obfuscation techniques, including code obfuscation, trivial obfuscation, renaming, reflection, encryption obfuscation, and combinations thereof. The AndroOBFS dataset is tailored to evaluate the model's robustness against obfuscated malware instances. It tests the model's ability to handle real-world complexities introduced by obfuscation techniques. For the Benign dataset we used the CCCS-CIC-AndMal-2020, [11][12], CICMalDroid2020 [13] and Androzoo [14] dataset which contains 3740, 1691 and 2600 benign samples respectively.

We opted for seven distinct machine-learning algorithms to serve as classifiers within our proposed model. The entire set of 167 extracted features was utilized in our experimentation. Training of the classifiers occurred using 70% of the labelled data, while the remaining 30% was allocated for testing purposes. To measure the effectiveness of the proposed model, five key evaluation metrics have been employed. 1)Precision (Pre): This metric quantifies the proportion of identified malware samples that are genuinely malicious, and in the context of benign samples, it indicates the proportion of identified benign samples that are truly non-malicious. 2) Recall (Rec): The recall metric measures the proportion of actual malware instances that were correctly identified as malicious. Similarly, for benign samples, it signifies the ratio of actual benign samples that were accurately recognized as non-malicious. 3) Accuracy (Acc): Accuracy denotes the ratio of correctly classified applications to the total number of applications, presenting an overall measure of classification correctness. 4) F-measure (F1): This metric represents the harmonic mean of precision and recall, thus offering a balanced evaluation of the model's ability to correctly classify both malicious and benign samples. It is calculated as 2*(Pre*Rec)/(Pre+Rec). 5)ROC curve: The area covered under the ROC curve is employed as an assessment metric. This curve illustrates the trade-off between the true positive rate and the false positive rate and is indicative of the model's ability to discriminate between different classes.

Table 1 Result: Classifying Malware and Benign Applications on Drebin and AndroOBFS

		Drebin				AndroOBFS			
Classifier	**Class**	**Pre**	**Rec**	**F1**	**Acc(%)**	**Pre**	**Rec**	**F1**	**Acc(%)**
SVM RBF	Malware	0.98	1.00	0.99	99.11	1.00	0.97	0.98	97.72
	Benign	1.00	0.99	0.99		0.94	0.98	0.96	
Naive Bayes	Malware	0.93	0.91	0.92	92.85	0.99	0.97	0.98	97.36
	Benign	0.93	0.94	0.93		0.94	0.98	0.96	
KNN	Malware	0.99	0.97	0.98	98.11	0.99	0.98	0.99	98.30
	Benign	0.97	0.99	0.98		0.96	0.99	0.98	
Decision Tree	Malware	0.97	0.97	0.97	97.31	0.99	0.99	0.99	98.50
	Benign	0.98	0.97	0.97		0.98	0.98	0.98	
Random Forest	Malware	0.98	0.99	0.98	98.55	1.00	0.99	0.99	99.27
	Benign	0.99	0.98	0.99		0.99	0.99	0.99	
ADA Boost	**Malware**	**1.00**	**1.00**	**1.00**	**99.40**	**0.99**	**1.00**	**0.99**	**99.39**
	Benign	**1.00**	**1.00**	**1.00**		**1.00**	**0.99**	**1.00**	

4.2 RQ1: What is the performance of the proposed model on classifying malware and benign applications on obfuscated and non-obfuscated malwares?

The task of malware classification involves discerning malicious samples from benign ones. We employed the Drebin dataset for non-obfuscated malware samples and the AndroOBFS dataset for obfuscated malware samples. CCCS-CIC-AndMal-2020 and CICMalDroid2020 datasets are used as the source of benign samples. This arrangement allowed us to assess the performance of our proposed model, leveraging seven distinct classifiers, in detecting malware.

To assess the resilience and versatility of our model, we turned to an obfuscated malware dataset. Table 1 illustrates the performance of the model in detecting non-obfuscated as well as obfuscated malwares. The data presented in Table 1 shows that, in the case of the Drebin dataset, the AdaBoost classifier achieved the highest accuracy score of 99.4%. This outcome reinforces the efficacy of our model in dealing with non-obfuscated malware samples, as the Drebin dataset primarily comprises non-obfuscated samples. The same classifier also demonstrated the best performance on the AndroOBFS dataset with an accuracy of 99.39%. AndroOBFS dataset contains obfuscated malware samples and this substantiates the model's fine performance even in the presence of obfuscation, further validating its versatility and robustness across varied scenarios. The confusion matrix of AdaBoost classifier's result with the Drebin dataset is shown in Figure 3.

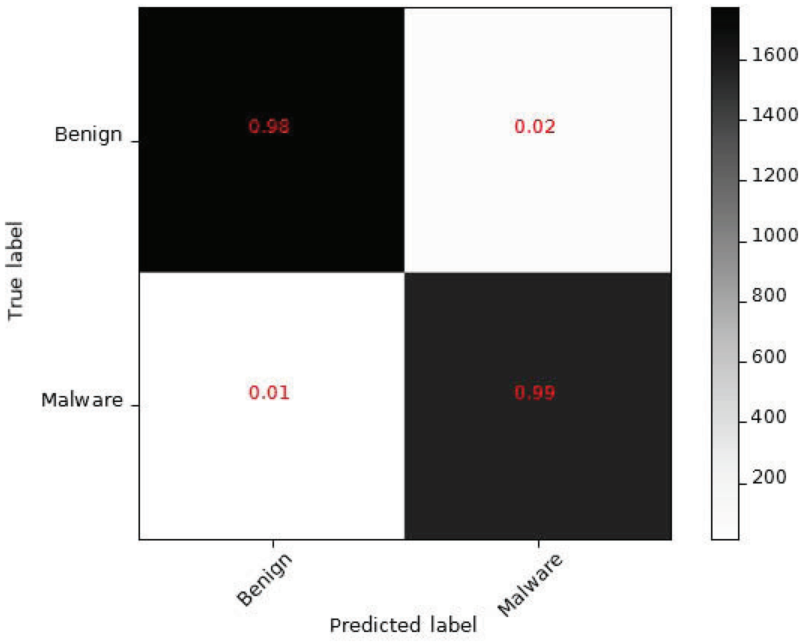


Fig. 3 Confusion Matrix of Malware and Benign Classification

Table 2 Results: Classifying Obfuscated and Non-obfuscated Malware on AndroOBFS and Drebin datasets

Classifier	**Class**	**Pre**	**Rec**	**F1**	**Acc(%)**
SVM RBF	Obfuscated Non-Obfuscated	1.00 0.97	0.97 1.00	0.98 0.98	98.41
Naive Bayes	Obfuscated Non-Obfuscated	0.82 0.96	0.96 0.81	0.89 0.88	88.33
KNN	Obfuscated Non-Obfuscated	0.99 0.99	0.99 0.99	0.99 0.99	98.39
Decision Tree	Obfuscated Non-Obfuscated	0.99 0.99	0.99 0.99	0.99 0.99	99.30
ADA Boost	Obfuscated Non-Obfuscated	1.00 0.99	0.99 1.00	1.00 1.00	99.68
Random Forest	**Obfuscated** **Non-Obfuscated**	**1.00** **1.00**	**0.99** **1.00**	**1.00** **1.00**	**99.74**

4.3 RQ2: What is the performance of the proposed model on classifying obfuscated and non-obfuscated malwares?

Detecting obfuscation within malware holds significant importance, as awareness of its presence is pivotal for effectively dealing with obfuscated malicious instances. Our proposed model takes this aspect into account by distinguishing between obfuscated and non-obfuscated malwares. The Drebin dataset is used for training and testing

Table 3 Results: Classifying Obfuscation techniques on Malware (AndroOBFS) dataset

Classifier		**Class**					
		Code	**Encryption**	**Reflection**	**Renaming**	**Trivial**	**Mix**
SVM RBF	Pre	0.98	0.71	0.99	0.40	0.00	0.83
	Rec	0.91	0.69	0.59	0.68	0.00	0.99
	F1	0.95	0.70	0.74	0.51	0.00	0.91
	Acc(%)			79.19			
Naive Bayes	Pre	0.97	0.45	0.10	0.30	0.08	0.68
	Rec	0.80	0.07	0.89	0.04	0.07	0.97
	F1	0.88	0.13	0.19	0.07	0.08	0.80
	Acc(%)			53.12			
KNN	Pre	0.98	0.77	0.85	0.46	0.14	0.98
	Rec	0.95	0.74	0.82	0.70	0.01	0.97
	F1	0.96	0.75	0.83	0.55	0.01	0.98
	Acc(%)			83.07			
Decision Tree	Pre	0.98	0.83	0.87	0.47	0.16	0.99
	Rec	0.95	0.70	0.89	0.76	0.03	0.99
	F1	0.96	0.76	0.88	0.58	0.06	0.99
	Acc(%)			84.00			
AdaBoost	Pre	0.88	0.53	0.29	0.35	0.04	0.83
	Rec	0.88	0.33	0.77	0.23	0.10	0.77
	F1	0.88	0.42	0.43	0.28	0.05	0.80
	Acc(%)			63.15			
Random Forest	**Pre**	**0.98**	**0.83**	**0.94**	**0.47**	**0.15**	**0.99**
	Rec	**0.96**	**0.71**	**0.89**	**0.77**	**0.03**	**0.98**
	F1	**0.97**	**0.77**	**0.91**	**0.59**	**0.05**	**0.99**
	Acc(%)			**84.35**			

Table 4 Results: Family classification performance of the proposed model with Random Forest for Drebin Dataset(top 10 families)

Family	**Count**	**Pre**	**Rec**	**F1**	**Acc(%)**
BaseBridge	330	0.95	0.90	0.92	96.80
DroidKungFu	667	0.96	0.97	0.96	
FakeDoc	132	1.00	1.00	1.00	
FakeInstaller	925	0.99	0.99	0.99	
Geinimi	92	1.00	0.83	0.90	
GinMaster	339	0.82	0.94	0.88	
Iconosys	152	0.98	1.00	0.99	
Kmin	147	1.00	1.00	1.00	
Opfake	613	0.99	0.99	0.99	
Plankton	625	0.99	0.94	0.97	

for non-obfuscated malwares while the AndroOBFS dataset is used for obfuscated malwares. Table 2 provides an overview of the evaluation metrics utilized to assess the model's diverse capabilities in recognizing obfuscated malwares amidst non-obfuscated ones. The findings in Table 2 highlight the model's performance in discerning whether a given malware sample has been obfuscated. The Random Forest classifier showcased

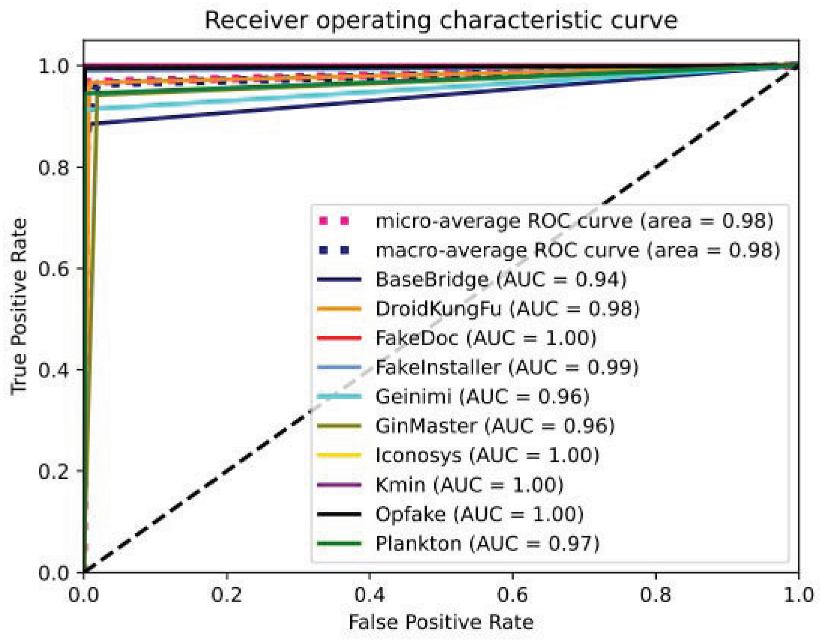


Fig. 4 ROC Curve for the familial classification

the best accuracy, achieving an accuracy of 99.74%. This outcome underscores the model's proficiency in effectively identifying obfuscated malware instances.

4.4 RQ3: How well proposed model effectively distinguish different obfuscation techniques employed on malicious samples?

In order to evaluate the proposed model's robustness in recognizing obfuscation techniques, we employed the AndroOBFS dataset. This dataset encompassed five distinct obfuscation techniques, namely code obfuscation, reflection obfuscation, renaming, encryption, and trivial obfuscation. Our objective was to gauge the model's efficacy in correctly identifying the various obfuscation methods applied to the malware instances. Table 3 highlights the evaluation metrics and the model's performance in discerning obfuscation techniques within the malware. Table 3 shows that the Random Forest classifier secured the highest accuracy of 84.35%. This classifier exhibited strong performance in terms of F1 Score, particularly when dealing with code, reflection, and mixed obfuscation, as the number of samples in these categories is quite healthy.

Table 5 Runtime overhead (in seconds) of the proposed model with sub-steps on 2000 Drebin malware samples

Steps	Total Runtime	Average Runtime
Reverse Engineering	27728	14.34
CSD Generation	125	0.062
Feature Vector Table Generation	412	0.20
Binary / Familial Classification	7.98	0.0039
Total	**28,272.98**	**14.60**

Table 6 Classification Accuracy of Proposed model and with existing state of art model on obfuscated and non-obfuscated datasets

Paper	Year	Features Used	Techniques	Dataset	Acc(%)
Karbab et al. [39]	2016	AndroidManifest, Permissions	Network addresses, APK	Drebin, Genome	87
Feizollah et al. [40]	2017	Permissions, Intent	Bayesian Network	Drebin, Google Play-Store	95.5
Wang et al. [41]	2017	Strings, Permissions, Intent, API calls	Random Forest, KNN, J48	Drebin, Genome Project	99.7
Garcia et al. [42]	2018	Intents, Permissions	SVM	Drebin, Virus Share, Malgenome and Virus Total	96
Alazab [43]	2020	API calls	KNN, RF, AdaBoost, Naive Bayes	VirusTotal, Mal-Share, AndroZoo	98.1
Sihag et al. [23]	2020	Opcode instructions	KNN, RF, J48	Drebin, Malgenome, PRAGuard	98.6
Millar et al. [44]	2020	Opcode, Permissions, API calls	CNN, DAN, Neural Network	Drebin and self obfuscated	97.3
Proposed work	**2023**	**Smali Opcodes**	**KNN, RF, AdaBoost, Naive Bayes, DT**	**Drebin, AndroOBFS**	**99.6**

4.5 RQ4: How well proposed model identify the malware families and classify them?

Familial classification involves categorizing malware samples into distinct malware families. To assess the effectiveness of the proposed model in family classification, we employed the Drebin dataset, which encompasses 179 diverse malware families. Our evaluation centred on the top 10 most prevalent malware families, totalling 4022 samples. Among the six classifiers utilized, the Random Forest classifier demonstrated superior performance. Table 4 outlines the outcomes of the familial classification conducted by the Random Forest classifier on the Drebin dataset. Random Forest achieved an accuracy of 96.80% while performing the familial classification. The ROC curve of this result is presented in Figure 4.

4.6 RQ5: What is the runtime overhead of the proposed model for classifying Android malware?

To assess the runtime overhead of the proposed model, a meticulous analysis was conducted on a subset of the Drebin dataset, comprising 2000 randomly selected samples. The model's operation involves four distinct stages for binary or familial classification of new malware samples. Each stage's runtime was evaluated as follows:

- Reverse Engineering: In this phase, the reverse engineering process involves decompiling the APK files of malware applications using APKtool. The runtime overhead of this process was calculated, revealing that the reverse engineering of the 2000-test sample took 27728 seconds as shown in Table 5. On average, APKtool required approximately 14.34 seconds to complete the reverse engineering of a single sample.
- CSD Generation: The second stage entails the generation of a Concatenated Smali Document (CSD) from the Smali files of decompiled applications. Using a Linux shell script, the process took 125 seconds to generate 200 CSDs. On average, it took about 0.062 seconds to create a single CSD as shown in Table 5.
- Feature Vector Table Generation: In the third step, a feature vector table is generated using the CSD and Dalvik opcode list. Table 5 shows the average runtime overhead for this process is approximately 0.20 seconds to generate a single row of the vector table. The generation of the feature vector table for all 2000 samples took 412 seconds.
- Binary / Familial Classification: The final classification step involves the use of various classifiers (e.g., random forest, decision tree, SVM-RCF, AdaBoost) to classify malware and benign applications, along with familial classification. The combined training and testing stage for classification took 7.89 seconds for the 2000 test samples. On average, it required about 0.0039 seconds to classify a single sample as shown in Table 5.

Overall, this detailed analysis provides insights into the runtime overhead associated with each stage of the proposed model's operation.

5 Discussion

The major finding of this research centres around the innovative utilization of Smali opcodes harnessed through the Dalvik opcode list and used in diverse machine-learning classifiers with the core objective of effectively classifying benign and malware samples. Significantly, the research also concentrates on the classification of obfuscated malware and benign samples, leading to several pivotal discoveries. The AdaBoost classifier emerges as the best performer in the binary classification of obfuscated and non-obfuscated malware alongside benign samples. AdaBoost classifier due to its adaptive nature, combines weak learners into a weighted sum and subsequently boosts the final classification outcome. This characteristic inherently suits binary classification tasks, contributing to its favourable results.

The research further finds the model's ability to identify the presence of obfuscation in malware samples. Employing the Drebin dataset as a non-obfuscated benchmark and the AndroOBFS dataset as an obfuscated counterpart, AdaBoost once again demonstrates commendable performance, achieving 99.68% accuracy. However, Random Forest slightly surpasses AdaBoost, boasting an accuracy of 99.74%. Random Forest's excellence is anchored in its construction of a multitude of decision trees during training, with the final classification output based on the class most frequently chosen by the most trees. This property empowers Random Forest to excel not only in binary classification scenarios but also in multi-class and diverse classification tasks. The research extends to the classification of various obfuscation techniques within malware samples. This encompasses the identification of five distinct obfuscation methods, along with a mixed obfuscation technique. Random Forest once again emerges as the dominant classifier, achieving an accuracy rate of 84.35%. This can be attributed to Random Forest's proficiency in constructing decision trees for diverse classification tasks, contributing to its ability to identify obfuscation techniques. Familial classification is another finding in the research, wherein Random Forest continues its better performance. The Drebin dataset is used for this experiment, the model successfully classifies the top ten most populous malware families, securing an accuracy of 96.80%. The research's final finding centres on the runtime overhead associated with the proposed model in the context of malware analysis. The experiment was conducted, utilizing 2000 Drebin malware samples and executed on an Intel(R) Core(TM) i7-1165GH processor clocked at 2.80 GHz, coupled with 8GB of RAM. The process comprised five sub-steps, as illustrated in Table 5 with the highest overhead of the reverse engineering step, which incurred a significant impact on the overall process. This step often results in the generation of numerous unnecessary directories and subdirectories, contributing to the elevated overhead. Future efforts could be directed toward optimizing this reverse engineering process, thereby streamlining the runtime performance. The remaining four sub-steps demonstrated favourable runtime performance, reflecting minimal overhead. This outcome underlines the effectiveness of these sub-steps within the proposed model, contributing to efficient execution times.

A thorough comparison with state-of-the-art malware detection models is presented in Table 6. The comparison highlights the superior performance and efficiency of the proposed model across various scenarios, particularly when dealing with obfuscated and non-obfuscated datasets. In comparison to existing state-of-the-art approaches in malware classification, our proposed model demonstrates superior accuracy performance, except for the work by Wang et al. In their study, Wang et al. utilize a comprehensive set of features, including permissions, Intent, API calls, and strings, for feature generation. Although this approach yields better accuracy, it also introduces substantial preprocessing and execution overheads. Furthermore, Wang et al.[41] employ the Genome project dataset in conjunction with the Drebin dataset, contributing to greater feature variation compared to the Drebin dataset alone. While their model achieves higher accuracy, it does so at the cost of increased computational complexity.

Our proposed model also exhibits robustness in identifying obfuscation techniques, achieving an impressive accuracy rate of 84.35%. This evaluation is conducted using the AndroOBFS dataset, which contains five distinct obfuscation techniques: code, encryption, reflection, renaming, and trivial obfuscation. However, the unequal distribution of these techniques within the dataset impacts overall performance, particularly for obfuscation types with smaller sample sizes. The dataset comprises 6895 code-obfuscated samples, 3436 encryption-obfuscated samples, 747 reflection-obfuscated samples, 1656 renaming-obfuscated samples, 512 trivial-obfuscated samples, and 1017 samples utilizing a mixture of the aforementioned techniques. Consequently, Table 3 illustrates high precision, recall, and F1 scores for code and encryption obfuscation due to their larger sample sizes. Conversely, the F1 scores for the remaining obfuscation types are lower due to their limited sample sizes, thus affecting the model's overall accuracy in identifying the type of obfuscation.

While the proposed model exhibits promising performance and efficiency, certain limitations are noted. Notably, the average runtime overhead during reverse engineering remains relatively high, suggesting room for improvement through alternative methodologies. The model effectively distinguishes the presence of obfuscation in malware instances, yet there's potential for enhancement in identifying specific obfuscation types. The same holds for familial classification, where the limited malware sample size impacts model training. Future work could involve utilizing larger datasets for training and refining the model accordingly. In summary, the research showcases the adept utilization of Smali opcodes for accurate malware classification, obfuscation detection, and familial classification.

6 Conclusion

The static malware analysis and its subsequent classification presents a complex challenge, involving the recognition and analysis of various features derived from the APK files of applications. This complexity is further increased with the inclusion of obfuscation in malware instances. The introduction of different obfuscation techniques fails the traditional classification and analysis methodologies to capture the distinguishing features responsible for malware identification. This paper introduces a novel model based on CSD (Concat Smali Document) and Dalvik opcode list for malware detection and familial classification. The model is effective, resilient and accurate most of the time when it comes to obfuscated malwares. The model's evaluation was conducted on two distinct malware datasets, encompassing both obfuscated and non-obfuscated malware samples. The evaluation aimed to address binary classification, familial classification, obfuscation type detection, and binary classification within obfuscated malwares. The datasets employed included Drebin and AndroOBFS malware datasets, coupled with the CCCS-CIC-AndMal-2020 and CICMalDroid2020 datasets for benign samples.

In future work, there lies the potential to extend the exploration beyond DEX and Smali files, encompassing elements such as library and resource directories. Additionally, the consideration of more complex obfuscation techniques and larger datasets could serve as an intriguing aspect for future analysis and exploration.

7 Declarations

Availability of data and materials

Datasets used in paper are publicly available in Drebin [5], AndroOBFS [10], CCCS-CIC-AndMal-2020 [11] [12], CICMalDroid2020 [13]), Androzoo [14].

Competing interests
The authors declare that they have no competing interests.

Funding
No funding was received to support this research.

Acknowledgements
The authors sincerely acknowledge the support provided by the dataset providers for making their datasets publicly available.

References

[1] Suarez-Tangil, G., Stringhini, G.: Eight years of rider measurement in the Android malware ecosystem. IEEE Transactions on Dependable and Secure Computing **19**(1), 107–118 (2020)

[2] Mehrabi Koushki, M., AbuAlhaol, I., Raju, A.D., Zhou, Y., Giagone, R.S., Shengqiang, H.: On building machine learning pipelines for Android malware detection: a procedural survey of practices, challenges and opportunities. Cybersecurity **5**(1), 16 (2022)

[3] Dong, S., Li, M., Diao, W., Liu, X., Liu, J., Li, Z., Xu, F., Chen, K., Wang, X., Zhang, K.: Understanding Android obfuscation techniques: A large-scale investigation in the wild. In: Security and Privacy in Communication Networks: 14th International Conference, SecureComm 2018, Singapore, Singapore, August 8-10, 2018, Proceedings, Part I, pp. 172–192 (2018). Springer

[4] Hammad, M., Garcia, J., Malek, S.: A large-scale empirical study on the effects of code obfuscations on Android apps and anti-malware products. In: Proceedings of the 40th International Conference on Software Engineering, pp. 421–431 (2018)

[5] Arp, D., Spreitzenbarth, M., Hubner, M., Gascon, H., Rieck, K., Siemens, C.: Drebin: Effective and explainable detection of Android malware in your pocket. In: Ndss, vol. 14, pp. 23–26 (2014)

[6] Peng, H., Gates, C., Sarma, B., Li, N., Qi, Y., Potharaju, R., Nita-Rotaru, C., Molloy, I.: Using probabilistic generative models for ranking risks of Android apps. In: Proceedings of the 2012 ACM Conference on Computer and Communications Security, pp. 241–252 (2012)

[7] Wang, W., Wang, X., Feng, D., Liu, J., Han, Z., Zhang, X.: Exploring permission-induced risk in Android applications for malicious application detection. IEEE Transactions on Information Forensics and Security **9**(11), 1869–1882 (2014)

[8] Fan, M., Liu, J., Luo, X., Chen, K., Tian, Z., Zheng, Q., Liu, T.: Android malware familial classification and representative sample selection via frequent subgraph

analysis. IEEE Transactions on Information Forensics and Security **13**(8), 1890–1905 (2018)

[9] Fan, M., Luo, X., Liu, J., Wang, M., Nong, C., Zheng, Q., Liu, T.: Graph embedding based familial analysis of Android malware using unsupervised learning. In: 2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE), pp. 771–782 (2019). IEEE

[10] Kumar, S., Mishra, D., Panda, B., Shukla, S.K.: AndroOBFS: time-tagged obfuscated Android malware dataset with family information. In: Proceedings of the 19th International Conference on Mining Software Repositories, pp. 454–458 (2022)

[11] Keyes, D.S., Li, B., Kaur, G., Lashkari, A.H., Gagnon, F., Massicotte, F.: EntropLyzer: Android malware classification and characterization using entropy analysis of dynamic characteristics. In: 2021 Reconciling Data Analytics, Automation, Privacy, and Security: A Big Data Challenge (RDAAPS), pp. 1–12 (2021). IEEE

[12] Rahali, A., Lashkari, A.H., Kaur, G., Taheri, L., Gagnon, F., Massicotte, F.: Didroid: Android malware classification and characterization using deep image learning. In: 2020 The 10th International Conference on Communication and Network Security, pp. 70–82 (2020)

[13] Mahdavifar, S., Alhadidi, D., Ghorbani, A.A.: Effective and efficient hybrid Android malware classification using pseudo-label stacked auto-encoder. Journal of network and systems management **30**, 1–34 (2022)

[14] Allix, K., Bissyandé, T.F., Klein, J., Le Traon, Y.: Androzoo: Collecting millions of Android apps for the research community. In: Proceedings of the 13th International Conference on Mining Software Repositories, pp. 468–471 (2016)

[15] Kang, B., Yerima, S.Y., Sezer, S., McLaughlin, K.: N-gram opcode analysis for Android malware detection. arXiv preprint arXiv:1612.01445 (2016)

[16] Bakhshinejad, N., Hamzeh, A.: A new compression based method for Android malware detection using opcodes. In: 2017 Artificial Intelligence and Signal Processing Conference (AISP), pp. 256–261 (2017). IEEE

[17] McLaughlin, N., Rincon, J., Kang, B., Yerima, S., Miller, P., Sezer, S., Safaei, Y., Trickel, E., Zhao, Z., Doupé, A., *et al.*: Deep Android malware detection. In: Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy, pp. 301–308 (2017)

[18] Zegzhda, P., Zegzhda, D., Pavlenko, E., Ignatev, G.: Applying deep learning techniques for Android malware detection. In: Proceedings of the 11th International Conference on Security of Information and Networks, pp. 1–8 (2018)

[19] Chen, T., Mao, Q., Yang, Y., Lv, M., Zhu, J., et al.: Tinydroid: a lightweight and efficient model for Android malware detection and classification. Mobile information systems **2018** (2018)

[20] Sihag, V., Mitharwal, A., Vardhan, M., Singh, P.: Opcode n-gram based malware classification in Android. In: 2020 Fourth World Conference on Smart Trends in Systems, Security and Sustainability (WorldS4), pp. 645–650 (2020). IEEE

[21] Niu, W., Cao, R., Zhang, X., Ding, K., Zhang, K., Li, T.: Opcode-level function call graph based Android malware classification using deep learning. Sensors **20**(13), 3645 (2020)

[22] Iadarola, G., Casolare, R., Martinelli, F., Mercaldo, F., Peluso, C., Santone, A.: A semi-automated explainability-driven approach for malware analysis through deep learning. In: 2021 International Joint Conference on Neural Networks (IJCNN), pp. 1–8 (2021). IEEE

[23] Sihag, V., Vardhan, M., Singh, P.: BLADE: robust malware detection against obfuscation in Android. Forensic Science International: Digital Investigation **38**, 301176 (2021)

[24] Nivedha, K., Gandhi, I., Shibi, S., Nithesh, V., Ashwin, M.: Deep learning based static analysis of malwares in Android applications. Advances in Parallel Computing Technologies and Applications **40**, 133 (2021)

[25] Millar, S., McLaughlin, N., Rincon, J.M., Miller, P.: Multi-view deep learning for zero-day Android malware detection. Journal of Information Security and Applications **58**, 102718 (2021)

[26] Wei, F., Li, Y., Roy, S., Ou, X., Zhou, W.: Deep ground truth analysis of current malware. In: Detection of Intrusions and Malware, and Vulnerability Assessment: 14th International Conference, DIMVA 2017, Bonn, Germany, July 6-7, 2017, Proceedings 14, pp. 252–276 (2017). Springer

[27] Lajevardi, A.M., Parsa, S., Amiri, M.J.: Markhor: malware detection using fuzzy similarity of system call dependency sequences. Journal of Computer Virology and Hacking Techniques **18**(2), 81–90 (2022)

[28] Yang, C., Xu, J., Liang, S., Wu, Y., Wen, Y., Zhang, B., Meng, D.: DeepMal: maliciousness-preserving adversarial instruction learning against static malware detection. Cybersecurity **4**, 1–14 (2021)

[29] Khalid, S., Hussain, F.B.: Evaluating opcodes for detection of obfuscated Android malware. In: 2022 International Conference on Artificial Intelligence in Information and Communication (ICAIIC), pp. 044–049 (2022). IEEE

[30] Hashemi, H., Samie, M.E., Hamzeh, A.: Ifmd: image fusion for malware detection.

Journal of Computer Virology and Hacking Techniques **19**(2), 271–286 (2023)

[31] Chysi, A., Nikolopoulos, S.D., Polenakis, I.: Detection and classification of malicious software utilizing max-flows between system-call groups. Journal of Computer Virology and Hacking Techniques **19**(1), 97–123 (2023)

[32] Nguyen, H., Di Troia, F., Ishigaki, G., Stamp, M.: Generative adversarial networks and image-based malware classification. Journal of Computer Virology and Hacking Techniques, 1–17 (2023)

[33] Malhotra, V., Potika, K., Stamp, M.: A comparison of graph neural networks for malware classification. arXiv preprint arXiv:2303.12812 (2023)

[34] Qiu, J., Han, Q.-L., Luo, W., Pan, L., Nepal, S., Zhang, J., Xiang, Y.: Cyber code intelligence for Android malware detection. IEEE Transactions on Cybernetics **53**(1), 617–627 (2022)

[35] Manzil, H.H.R., Manohar Naik, S.: Android malware category detection using a novel feature vector-based machine learning model. Cybersecurity **6**(1), 6 (2023)

[36] Huang, W., Meng, G., Lin, C., Yan, Q., Chen, K., Ma, Z.: Are our clone detectors good enough? An empirical study of code effects by obfuscation. Cybersecurity **6**(1), 1–19 (2023)

[37] Puerta, J., Sanz, B.: Using Dalvik opcodes for malware detection on Android. Logic Journal of the IGPL **25**(6), 938–948 (2017)

[38] Winsniewski, R.: Android–apktool: A tool for reverse engineering Android apk files. Retrieved Febr **10**, 2020 (2012)

[39] Karbab, E.B., Debbabi, M., Derhab, A., Mouheb, D.: Cypider: building community-based cyber-defense infrastructure for Android malware detection. In: Proceedings of the 32nd Annual Conference on Computer Security Applications, pp. 348–362 (2016)

[40] Feizollah, A., Anuar, N.B., Salleh, R., Suarez-Tangil, G., Furnell, S.: Androdialysis: Analysis of Android intent effectiveness in malware detection. computers & security **65**, 121–134 (2017)

[41] Wang, X., Zhang, D., Su, X., Li, W., et al.: Mlifdect: Android malware detection based on parallel machine learning and information fusion. Security and Communication Networks **2017** (2017)

[42] Garcia, J., Hammad, M., Malek, S.: Lightweight, obfuscation-resilient detection and family identification of Android malware. ACM Transactions on Software Engineering and Methodology (TOSEM) **26**(3), 1–29 (2018)

[43] Alazab, M.: Automated malware detection in mobile app stores based on robust

feature generation. Electronics **9**(3), 435 (2020)

[44] Millar, S., McLaughlin, N., Rincon, J., Miller, P., Zhao, Z.: DANdroid: A multi-view discriminative adversarial network for obfuscated malware detection. In: Proceedings of the Tenth ACM Conference on Data and Application Security and Privacy, pp. 353–364 (2020)