International Journal of Safety and Security Engineering
Vol. 14, No. 2, April, 2024, pp. 513-521
Journal homepage: http://iieta.org/journals/ijsse

Enhanced Malware Detection for Mobile Operating Systems Using Machine Learning and Dynamic Analysis

Faris Mutar Mahdi Aledam[1*], Bilal Majeed Abdulridha Al-Latteef[2]

[1] Faculty of Administration and Economics, Al-Muthana University, Samawah 66001, Iraq
[2] The General Directorate of Qadisiyah Education, Computer System and Networks, Diwaniyah 58006, Iraq

Corresponding Author Email: Faris.mutar@mu.edu.iq

https://doi.org/10.18280/ijsse.140218

Received: 6 February 2024
Revised: 3 April 2024
Accepted: 11 April 2024
Available online: 26 April 2024

Keywords:
mobile smartphone, machine learning, principal component analysis (PCA), malware, android, iOS

ABSTRACT

Mobile smartphone operating systems have garnered widespread popularity due to their open-source nature and high performance. However, the convenience of these systems has also led to a rise in malware distribution. Traditional signature-based detection methods often fail to identify unknown threats, prompting the need for more effective solutions. In this study, we propose an advanced machine learning-based model for detecting malware on smartphones. Our model leverages dynamic and static analysis techniques to select and infer features, followed by a novel feature extraction method using sampling and Principal Component Analysis (PCA) to reduce dimensionality without adversely impacting the accuracy. Experimental results demonstrate the effectiveness of our approach in significantly enhancing malware detection accuracy and efficiency on smartphone operating systems. By analyzing the dynamic behavior of applications and incorporating innovative detection methods, our research contributes to a more robust and proactive approach to smartphone security. Through rigorous evaluation using real-world and synthetic datasets, we validate the efficacy of our model in accurately identifying malware instances and guiding users towards safe application downloads. Overall, our study provides a promising avenue for mitigating the escalating threat of malware on mobile devices.

1. INTRODUCTION

By the end of the decade, an array of technology-enabled cognition support tools emerged, transforming both manually based intelligence dispatching activities and routine workflows in several industries, organizations and government agencies [1]. This shift has been facilitated by the innovations that have had a far-reaching favorable impact on the acceptance of mobile technology at the global level. Based on UN report, it is expected that the percentage of people who carry the smartphones will reach 82% in 2022 [2].

Despite the fact that mobile technology has enhanced digital solutions for many workloads, it has also made consumers' data more vulnerable. Due to the lack of oversight in the Google Play Store, app developers are able to post Android programs with little to no filtering, increasing the likelihood of harmful apps being posted and endangering users' personal information and data [3]. Economic losses are another consequence of malware assaults on gadgets. The many hazards associated with mobile technology are not going unnoticed. Android is both the most popular and most susceptible mobile operating system [4]. Figure 1 shows that up to July 2023, Android OS accounted for more than 71.9% of the worldwide market in the mobile industry. Next on the list of mobile operating systems is iOS, which has around 27.3% of the worldwide market share as shown in Figure 2.

Android OS smartphones are widely used, making them a potential target for malware attacks. Another reason Android OS is vulnerable is because it is open-source [5, 6]. In 2022, 196,476 banking Trojans and 10,543 ransomware Trojans were anticipated to have been identified, according to the Kaspersky research [7]. Trojan mobile apps infiltrate the operating system by masquerading as genuine programs, while in reality, they are counterfeit. Banking Trojans allow customers to reveal their account information using phony banking applications, which is a problem since most consumers now utilize mobile Internet banking. In addition, users' health information and sensitive personal details are also disclosed. Numerous additional strategies, including blockchain technology and edge computing approaches, are now being used to safeguard such data [8]. Professionals in malware have compromised mobile devices and turned them into bots. Distributed denial-of-service (DDoS) assaults and spam emails with harmful links are both sent by these bots.

The development of this malicious software employs sophisticated techniques that make these assaults unavoidable [9]. These botnets seriously jeopardize the security of Android OS. Unfortunately, the Security Institute [10] reported that Android packages are a significant vector for malware attacks.

Therefore, signature-based methods of detecting malware and malicious installation packages utilizing attribute information may be efficiently used to improve Android

mobile security. The malware industry is likely teeming with specialists who are always thinking of new ways to conceal their assaults and steal sensitive user data. Neutralizing such novel and intricate approaches is, however, getting more difficult [11]. There are a lot of methods that have been developed to identify malicious activity in Android applications. These methods detect malware assaults and use feature interaction to show which features are important [12].

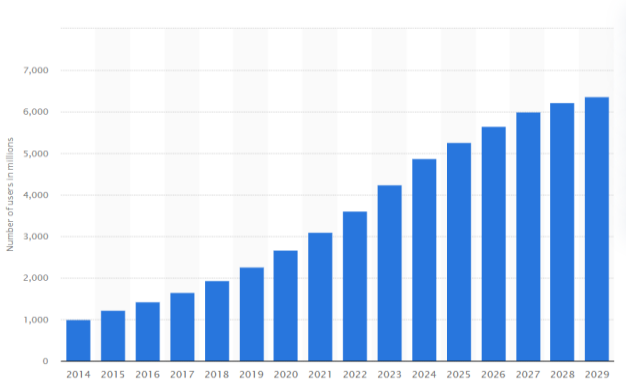

Figure 1. World smartphone smartphone usage between 2014-2029

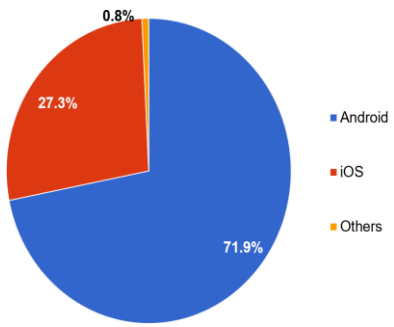

Figure 2. Statistics for the share of mobile operating systems from July 2022 to July 2023 [5]

For malware attack detection, the Owl binary optimization algorithm is used to select features from the Drebin dataset [13]. Using Android malware detection greatly increases security against any potential attack [14]. Furthermore, in recent years, many deep learning-based methods have been proposed to detect malware. For example, S.C. Tan et al. used back propagation (BP) and particle swarm optimization (PSO) to find the best ensemble classifier for deep learning. The ultimate objective of employing deep learning is to choose the most ideal attributes to improve the accuracy of malware detection systems, while simultaneously aiming to minimize the computational costs of computers. Detecting malware on mobile phones with high accuracy and ease of use is the goal of this research paper, which combines parallel machine learning classifiers and supervised algorithms. Also incorporated into the framework was optimal feature selection.

Modern machine learning networks use correlation scores to choose features, so you do not have to do any feature computations by hand before putting a deep learning classifier to work.

The contribution of the paper is evident in the following aspects:

A feature selection mechanism based on the correlation score is embedded in the machine learning network instead of performing a manual calculation of features before applying a deep learning classifier, which contributed to reducing the computational burden.

Without incurring an additional processing time requirement, the accuracy of malware detection was enhanced. Offers a cost-effective approach to detecting malicious or altered programs in mobile device operating systems. provides an alternative to malware detection of smartphone operating systems for malicious or recompiled applications at very low costs.

2. LITERATURE REVIEW

Researchers in malware detection analysis techniques are still constrained between two practical approaches. Strategies based on Android malware detection analysis are usually either static, dynamic, or hybrid. In this section, we review some analysis methods and briefly summarize their used properties.

Alabrah [13] presented a cutting-edge automated technique for detecting Android malware, based on artificial neural networks (ANN). To test this innovative method, two well-known datasets were utilized: CICInvestAndMal2019 and Drebin/AMD. These datasets underwent preprocessing to convert their static features into binary values, indicating the status of certain app permissions (enabled or disabled). The modified feature sets were fed into the ANN classifier for two crucial experiments. In the first experiment, a basic input layer was used alongside a five-fold cross-validation approach. For the second experiment, a novel feature selection layer was introduced in the ANN classifier, focusing on features correlated with benign or malware apps. The outcomes of Alabrah's ANN-based method were not only substantial but also showed enhancements in performance and resilience.

Tarwireyi et al. [14] introduced BarkDroid, a novel Android malware detection technique that uses the low-level Bark Frequency Cepstral Coefficients audio features to detect malware. The initial results obtained show that Bark Frequency Cepstral Coefficients have high discriminative capabilities to achieve accurate predictions.

In the study of Fan et al. [15], a method called free graphing was studied, in which sub-frequent graphs represent typical patterns from malicious systems that merge with the same package. They're also a template for FalDroid, which is a (free) chart-based detection system. Studies across multiple trials have shown that FalDroid can classify up to 96.3% of malicious system samples into their own divisions in about 6.2 seconds per app.

Fatima et al. [16] presented another model that works on a server-hosting basis to detect malicious systems. Through this approach, material costs can be reduced and resource constraints of more than 98% can be achieved, but the model needs high server-level specifications and features for immediate response time. In addition, this model did not discuss the information security involved in the process.

Cai and Jenkins [17] proposed a unique Android malware detection approach that, once tested on different categories of data, can effectively continue to detect new malware without retesting. Droid-evolver is a fully automated (without human intervention) system for detecting malicious apps for smartphone operating systems, automatically updating itself.

Fang et al. [18] used the feature fusion method and directly call the library function to extract the permissions and API

features of the APK file, then decompile the APK file to obtain the opcode features and merge the three features with multiple features to generate a feature vector. Finally, it uses a multi-model neural network HYDRA to learn fusion feature vector, so that it can identify and detect malware. The work also compared it with other single-feature machine learning algorithms to verify its effect. Experimental results show that the accuracy of the multi-model neural network detection method based on feature fusion reaches 98.92%, which is better than other single-model feature methods.

The composite method has been discussed in the study of Surendran et al. [19]. Through a script to detect malicious systems using the Bayesian Tree (TAN) model, which is based on dynamic and static features such as permissions and system calls, it detects the harmful model by combining the results of these two features taken from the segmentations. Moreover, the text shows 95%, but it does not show the smartphone OS version during the dynamic analysis. However, although the hybrid analysis method has been shown to be more complex and successful in the case of dynamic and static analysis, in the end, feature selection remains the key to the detection ratio.

The method used in the study of Al Ali et al. [20] was used to reach a detection ratio of 96. The compared the characteristics of dynamic analysis using integration, structure dimensions, and connectivity between components. They concluded that the specifications extracted using the hull dimensions were more significant than the other two.

3. PRINCIPAL COMPONENT ANALYSIS (PCA)

Principal component analysis (PCA) is a multivariate technique that analyzes a data table in which observations are described by several inter-correlated quantitative dependent variables. Its goal is to extract the important information from the statistical data to represent it as a set of new orthogonal variables called principal components, and to display the pattern of similarity between the observations and of the variables as points in spot maps [21, 22].

One of the most important features of PCA are:

Principal Components Analysis (PCA) aims to maximize the variance in the data by creating new axes called principal components. By selecting dimensions that capture most of the data's variance, PCA retains important information while reducing dimensionality, which is crucial in malware detection to maintain feature distinctiveness for accurate classification. PCA ensures that the new axes are orthogonal, meaning each component captures a unique aspect of the data's variability, resulting in a more concise representation of the original features. Unlike other dimensionality reduction methods, PCA offers consistency and minimizes redundancy or information loss. While PCA assumes linear correlations among variables, which may not always hold true, it is generally effective in capturing the underlying data structure without significant loss. Its computational efficiency and ease of use make PCA a preferred choice for handling large datasets in malware detection studies compared to methods like t-SNE or Isomap.

PCA delivers a coherent interpretation of the condensed feature space via principal components, which represent linear combinations of the original features. This interpretive capability aids in analyzing feature significance and enhances comprehension of the intrinsic data structure.

4. DYNAMIC AND STATIC ANALYSES

To identify malware, almost absolute majority of static statistical methods and dynamical approaches are applied [23, 24]. Two detection approaches are available to the user with static analysis: heuristic analysis, and signature-based detection technique There are two different techniques in antivirus software, which are very arguable among programmers. As the signatures can only look for the patterns of the know malware, they are no longer the ultimate way to achieve the entire security. What is a contrast between the both scenarios is in place. In the first scanner, it identifies risks according to its specific purpose which is to spy malicious files that are programs and deliver warnings when they are noticed.

Through the study of code’s traits and/or the way how the form behaves. Users may consider code analysis as an applicable option. Code structure examination involves finding malicious code patterns by looking at the syntax of the code as well as picking out how it is arranged. Alternatively, string analysis entails activities such source code inspection even for signs of malicious intent such as IP addresses, encryption keys, or hardcoded URLs. During data analysis, properties of files, like the size, creation date, and digital signature have to be observed to find alterations that are unlawful or might look like certain type of damage. An additional method, that completes the analysis of the execution code can find malicious and inappropriate behavior, like including concealed functions and code obfuscation.

While sandboxing analysis means executing the binary in a simulated environment for the purpose of seeing the trait behaviors, code interactions with the system, and detection of any network traffic that’s sniffy or suspicious activity.

The most prevalent dynamic analysis technique, similar to static analysis, encompasses [25, 26]:

Runtime behavior appraisal is a process of checking the activity of code or files to detect code(s) or files(s) that have questionable or true malicious behaviors.

This includes, for instance, the unlawful revamping of the system, the modifially of the file system or the monutring of the network. API monitoring is a method of observation that focuses on how the codes call the "Application Programming Interfaces" (APIs) in order to detect any suspicious or severe calls that are likely to be attempts to break in and cause damage. Network traffic analysis is equivalent to taking a document and reviewing the network traffic that may be related to the operation of the code or the file itself. The targeted objective is to trace and flag any transaction with confirmed fraudulent websites, abnormal data transportation, or abnormally high network activity. In contrast to static code analysis modality which deals with looking closely at the written code for any statistic malicious operations, dynamic code analysis extends the objective to include thorough processes to point out suspicious and hazardous experiences while the code is running. System call monitoring implies monitoring system calls of applications, and files made to the operating system in order to stipulate malicious actions followed by the abnormal or illegal system call. Sandboxing is a way that programs or files are run in a virtual environment (sandbox) to observe its reactions and maintain it separated from other entities, thereby spoiling any possible harm in the main system.

The emulator and virtual machines reproduce runtime environment of the target system in which the code or file is executed. By this approach the analysts can see the behavior of the code and its interactions with other applications /

components without affecting the host system. Implementing these tools along allows to identify and categorize malicious and plays an important role in the security of systems and networks.

5. PROBLEM FORMULATION

In the protection phase, developers provide a trained model for users to detect malware where the software is able to independently reach a decision based on system predictions as shown in Figure 3. Errors can lead to great risks for the user - such as removing the phone's operating system. It is necessary for the developer to choose a model family correctly. The developer should use a robust affirmative training procedure to achieve the ideal model with a high detection rate and effective positive rate.

User machines that apply machine learning models make decisions on their own. The quality of the machine learning model affects the functioning of the user's system. For this reason, machine learning-based malware detection has specific characteristics.

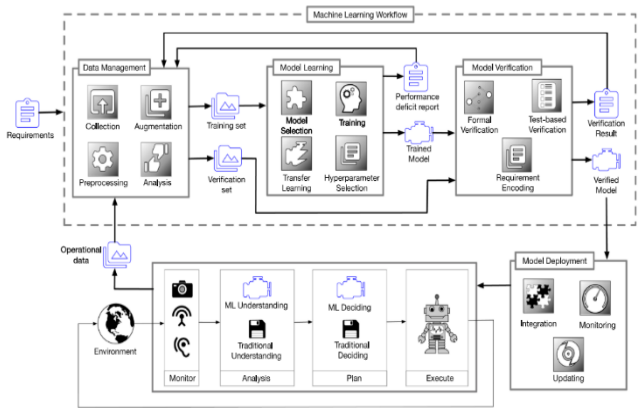

Figure 3. Detection algorithm lifecycle using machine learning [27]

Sample selection is based on the Zone-Alarm suite of applications (for security applications). Originally, a batch of 270 good apps and 270 others with malicious behavior was made to try to cover up a certain randomness.

The collection of cute apps has been chosen to try to be diverse and reflect the different types of apps on the Play Store app. It is also proportional to the number of samples present in each type of application [28]. The following aspects were taken into account when collecting mock samples:

a. Similar and different sized apps with the same name and malware variants.

b. Different classifications according to the behavior of those with the greatest impact: SMS Trojans, banks, root extortionists, extortionists and criminals, adware and malicious tools.

C. Different transactions within one malware package and more than one package within a classification by pattern.

For the analysis, 6227 samples were selected from the same repository, of which 4105 were infected and 2122 benign. In addition, for the set of samples with detrimental behavior, at least one variant from each bundle was detected in the system. Drebin contains 7,220 samples of infected software owned by 319 malware packages. For the detection of recombinant infected programs, 1912 samples of infected programs were selected from the top 4 packages with the number of samples in each package (Table 1) [29]. In addition, the specific software has been changed to have multiple features such as permissions and package names. Surprisingly, many duplicates were found among the package names of applications after analysis. It was concluded that about 68.19% of the applications in the dataset share a number of repeated package names, and therefore, the applications that share the same package names were sorted.

The orderly compilation of smartphone operating system applications can make a positive or negative change in the application signature. Because of this, all applications that share the same package names continue to have different hash values, and therefore, it was necessary to create a more robust signature technology. The primary goal in this part of the study is to update an efficient signature mechanism so that about 95% of the samples with package names have identical signatures. Then the hash of the class.dex file is developed for all the open-source code obtained from the application, instead of using the hash value calculation method [30].

A detailed report of applications shared in family names shows that 90% of them use the same source code with minor changes. Hash algorithms, such as SHA-115 and MD-516, load from a file of random size and a fixed-length cryptographic hash as a result. Computing a SHA-1 or MD-5 hash for two identical files will most of the time yield the same result. Antivirus software stores up-to-date databases of MD-5 and SHA-1 hashes of malware. In addition, a small modification to the infected system causes a very large change in the SHA-1 or MD-5 hashing process.

Table 1. Malware samples in the Drebin dataset from the top 4 packages

Malware Packet	Samples
Kmin	147
FakeDoc	132
FakeInstaller	821
OpFake	363

Therefore, a new, more efficient hashing technique called SSDeep hash was used, instead of calculating the source codes of applications that share the same package names by SHA-1 or MD-5 algorithms. SSDeep is based on context driven segmentation (CTPH) technology known as fuzzy segmentation. CTPH is a new technology that improves the effectiveness of similar file detection. Because of the fuzzy hashes of two highly identical files, i.e. the original file and a file with some minor changes, SSDeep hashes can give the degree of similarity between two hashes. If there are any minor degree changes in the cloned software and malware, a similarity score can be obtained by comparing it to the malware which is the ability to compare the similarity between two algorithms [31].

Algorithm 1: Detect malware repackaged using Fuzzy hash
Input: FH = {h1, h2, h3…….hn} and APK
Output: Similarity-Score
1: hash SSDeepHash (APK)
2: for all i € FH do
3: Similarity SSDeepSim (i, hash)
4: if Similarity > threshold then
5: Return Similarity
6: end if
7: end for

8: Return 0

Algorithm 1 introduces a new approach based on fuzzy hashing to detect repackaged malware. We assume that F is the set of the top 4 bundles of the Drebin data set. We do the reverse process of designing all applications in F to choose a bunch of distinct package names such as DPN = {Pn1, Pn2, Pn3, Pnn}. In addition, one application model is randomized for each package name in the DPN, after that, we do a mathematical operation to calculate the fuzzy hash using SSDeep and put it into the FH matrix. In the end, a fuzzy hash package of FH and an APK of F are obtained as input in Algorithm 1, while the similarity ratio is chosen as the final result.

First: Get the fuzzy hash value of the parent code of the given APK (step 1).

Second: We compare the algorithm of the APK file with the whole algorithm in FH with the help of SSDeep hash comparison tool (step 3).

Third: If there is a similarity ratio greater than the threshold value at any point, the APK file will be marked as a recompiled infected program, and the similarity score will be returned again. (Step 4-6).

Fourth: The value of zero is returned to the algorithm if there is no similarity ratio higher than the minimum hash in FH. A similarity ratio of 85% was set for the standard cut-off for the trials [32].

As mentioned earlier, the technologies for detecting malicious APK files are divided into dynamic and static features. Dynamic analysis works with the pattern of the running time of the programs at the time of their execution compared to several specific experiments. Although the hard side of the analysis is done at a non-running stage (as opposed to) in terms of verifying the source code, analyzing metadata and additional data about vulnerabilities. Dynamic analysis is an accurate detection method because it involves detailed analysis of applications, so it requires a high amount of money. After executing the APK files, the analysis is performed [33, 34].

Static analysis consists of a very large set of techniques and methods that aim to learn about the patterns and behavior of the system runtime before implementing it. The main goal of increasing security is to separate applications that will be recompiled from malware before execution and installation processes [35].

6. METHODOLOGY

In this section, we discuss our approach to developing a malware detection model based on the analysis of effective and early system calls coupled with evaluation by an application.

The model proposed in the study of Zhu et al. [33]. Detection percentage with reduced passes based on enhancing validity features. As we conclude from the Per-DRaML detection system based on the proposed scheme using permissions from the applications themselves and their applications, Per-DRaML targets a set of specific permissions enhanced in improving the percentage of detection of dangerous programs, rather than analyzing all required permissions. Random Forest algorithms, Support Vector Machine (SVM) and Rotation Forest classifiers were used for classification. Based on the perceived effect on the detection effectiveness of systems and malware, we will select a set of powers. We will discuss some important issues in this paper:

1. Packets of benign and malignant specimens.
2. Build/define the feature set.
3. Key Features (dataset) Inference, Filter and Finalize.
4. Classification of Android malware using moderated eLearning algorithms.

6.1 Packets of benign and malignant specimens

A set of Android applications has been selected from two different groups of android families, benign and malicious. Virus-Share (about 7,000) malicious apps have been aggregated into an Android malware database (http://virusshare.com/, December 25, 2022). Virus-Share's database identifies application packages from different malware packages at different dates and is available to all as archived and compressed files. These files can be obtained using any torrent's user. A bunch of benign apps (about 7000) were also selected from the official app site (Google Play and Apple Store) using the Python language implementation. Innocent APKs are selected from different Play Store app ratings to increase diversity in the dataset. The total APK files are 14,000 samples, each classification has 7,000 samples. Training data is used to evaluate the effectiveness of the current model, while samples are used to perform validations.

6.2 Build/define the feature set

In the first stage, classifier schemas are built and classified in the selection of key permissions based on the data set. The permissions and features required by the app are obtained in the form of an app package such as: APK and Manifest.xml files. To obtain the required validity, the Andro-guard algorithm is adopted to unpack 14,000 application samples for the required data bundle. Different classifications of permissions used to create the feature set package, such as small application sizes and permission ratio, have been selected to perform consistent analysis and understanding of the style of each application from the selected packages [34].

6.3 Key Features (dataset) inference, filter and finalize

This pane shows the most important permissions that can be used to separate apps from benign and malicious apps. Google Systems and Zhu et al. [33] were able to extract the list of dangerous permissions. To identify the main permissions important for malware detection, several permissions, as shown in Table 2, were presented as illustrative samples. It is noted from the table that Zhu et al. Google permissions are integrated to be evaluated while using an exclusive feature called Permission Ratio [35, 36].

Figure 4 shows the proposed Per-DRaML model, which demonstrates filtering of APKs parameter specification for a dataset and packages, de-compilation and refactoring.

6.3.1 Enhanced permissions package

First the permissions that have a weak impact on detection are obtained to determine the minimum value of the number of permissions required. To this end, we used a dataset from Google's permissions list (from Table 2) documenting the functionality and importance of the feature. Feature significance is the metric that leads to the creation of simpler, more efficient prediction recipes using less data. When the feature significance of the Random Forest model is used, some

significant powers are shown (Table 2). We set a threshold standard of 0.7 to choose the feature that has the most impact by avoiding permissions that have a significance of less than 0.7. Where the most important validity models were identified, based on a set of different feature set samples. As shown in Table 3.

Table 2. Permissions risky feature and its significance by Google - G and Zhu et al. - R with exceptional standards

S. No.	Features	Importance %	Source
1	READ_PHONE_STATE	0.42674	G
2	Permission rate	0.30376	R
3	WRITE_EXTERNAL_STORAGE	0.07948	G + R
4	ACCESS_ APPROXIMATE _LOCATION	0.06323	G
5	RECORD_AUDIO	0.02923	G
6	READ_EXTERNAL_STORAGE	0.02588	G
7	CAMERA	0.02024	G
8	RECEIVE_SMS	0.02019	G + R
9	READ_SMS	0.00843	G
10	READ_ADDRESS	0.00650	G
11	WRITE_CALL_LOG	0.00079	G
12	UPDATE_DEVICE_STATS	0.00016	G + R
13	READ_HISTORY_BOOKMARKS	0.00003	G + R
14	WRITE_HISTORY_BOOKMARKS	0.00000	G + R

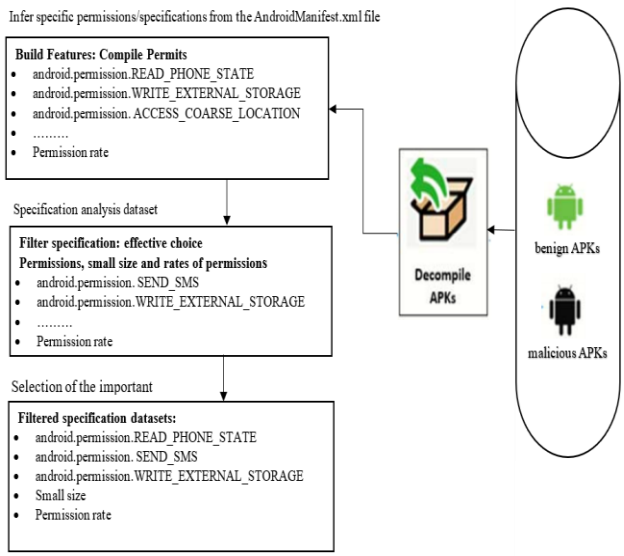

Figure 4. Diagram of specification generation, dataset generation, and filtering for malicious and benign APKs

Table 3. Proposed models of the features of the specified parameters

Type	No.	Name
Permissions	1	Android. Permission. WRITE_EXTERNAL_STORAGE
	2	Android. Permission. READ_PHONE_STATE
	3	Android. Permission. ACCESS_ APPROXIMATE _LOCATION
Standards	1	Small Size
	2	percentage validity

6.3.2 Designed dataset

Permission packets are converted into a binary dataset so that '1' is the program granting validity, and '0' denotes no validity. Permission models selected from a few benign and malicious applications, represented binary, are combined to design a single comprehensive dataset for analysis.

6.4 Classification of android malware using moderated machine learning algorithms

Supervised machine learning assessments were used in this part, which can detect dangerous programs with the least amount of positive error value. The general plan of the current model is divided into two categories; The first consists of a standard in which supervised trainees are trained and validated using datasets with different machine learning algorithms, and the second category is validity feature inference. As mentioned earlier, the data set used consists of 14,000 samples consisting of 7,000 samples of each type. A similar training method and testing algorithm was applied in experiments as Zhu et al. [33, 35].

7. PERFORMANCE EVALUATION

Standard evaluation criteria are described: accuracy, sensitivity and Receiver Operating Characteristic (ROC) curve. Where we review later the formulas and their definitions [36].

Confusion matrix consisting of four criteria are: true positive (TP), true negative (TN), false positive (FP), and false negative (FN).

To analyze the results of the model framework used, we used the following criteria:

1) Accuracy: This is the percentage of correctly selected APKs.

$$\text{Accuracy} = \frac{TN + TP}{FP + FN + \text{TP} + TN} \tag{1}$$

2) Precision: is the number of correctly predicted phishing APKs.

$$\text{Precision} = \frac{TP}{TP + FP} \tag{2}$$

3) Recall: is the collection of phishing APKs that have been segmented and validated.

$$\text{Recall} = \frac{TP}{TP + FN} \tag{3}$$

4) F scale: is the weighted harmonic average of test accuracy and recall. At value 1 it will be positive and at value 0 it will be negative.

$$\text{F1} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{4}$$

We trained our model using 4 phases, and reported the results for each phase as shown in Table 4.

The standard of performance of malware detection systems can be increased either by improving certain powers and features or by improving data collection. Where the high performance is to choose important permissions of the

proposed method, they are influential figures considered in the literature. The application needs to obtain the permission of the user to perform the necessary activities. The proposed model aims to evaluate performance in the careful selection of data set samples. Additionally, this model was trained and validated on a large dataset, around 14,000 APK files were used as samples, obtained from places as diverse as virus sharing and application web sites. It has recently been observed that the current model achieves a similar detection ratio in the Rotation Forest and SVM algorithms using the given current powers, when compared to the results obtained from standard methods as shown in Table 3. To achieve high detection accuracy can Classifiers help reduce the number of batches being reserved, furthermore, reduce computational overhead, and can become a cost-effective solution for malware detection [33].

Table 4. The results of the training phase over several time stages

Stage	Time	Accuracy	Loss
1	189s	0.8442	0.3562
2	188s	0.8889	0.2709
3	186s	0.9012	0.2415
4	184s	0.9151	0.2104

Figure 2 shows the percentage of malware data packets that fall within the classifications shown.

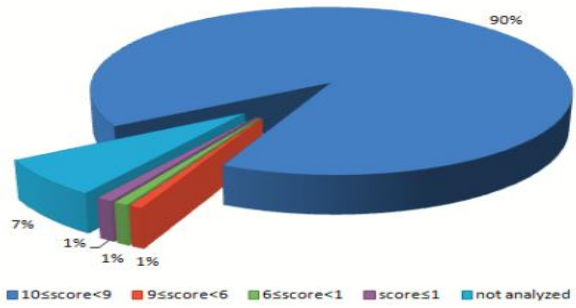

Figure 5. Percentage ratings of malware samples

The Figure 5 shows that 90% of the malware samples were effectively identified as malware (6,132/ 6,648) in the fourth type (9≤score≤10), and the malicious type almost reached 1% with the use of 52 samples belongs to this type. Regarding the latter two types, only 1% of the samples were rated as almost reliable (52/6075) and another 1% as almost reliable (67/630). 582 files, i.e. 7% of the data packets (391/5,560) were not parsed. 1210 files of type IV (malware type, 9≤score≤10) were given a score equal to 10, which is the highest end of the malware classification according to the Andrubis study [32].

8. CONCLUSIONS

While mobile malware continues to pose a persistent threat to Android users, the increasing integration of smartphones into our daily lives underscores the critical need for robust security measures. Therefore, the development of novel and effective malware detection technologies should be prioritized.

In this study, we evaluated various metrics and criteria, including malware detection rates, resource utilization, machine learning schemes, and extracted models for analysis, to assess the efficacy of malware detection technologies. We compared and analyzed techniques and models from previous research, considering factors such as unknown malware detection, which was not part of the training set.

Our approach involved a multilevel model, where we initially identified and inferred significant features from a dataset comprising 14,000 application samples. We utilized various machine learning frameworks to classify applications as benign or harmful. Through a series of experiments, our proposed model demonstrated significant enhancements in predictive features and the identification of harmful applications.

Moreover, our model offers a cost-effective alternative for detecting malware in smartphone operating systems, particularly malicious or recompiled applications. However, it is essential to acknowledge the limitations of our research, such as the need for further investigation into addressing unknown malware detection and refining the feature selection process.

REFERENCES

[1] Bai, H.P., Xie, N.N., Di, X.Q., Ye, Q. (2020). Famd: A fast multifeature Android malware detection framework, design, and implementation. IEEE Access, 8: 194729-194740. https://doi.org/10.1109/ACCESS.2020.3033026

[2] Atacak, I. (2023). An ensemble approach based on fuzzy logic using machine learning classifiers for android malware detection. Applied Sciences, 13(3): 1484. https://doi.org/10.3390/app13031484

[3] Chopra, R., Acharya, S., Rawat, U., Bhatnagar, R. (2023). An energy efficient, robust, sustainable, and low computational cost method for mobile malware detection. Applied Computational Intelligence and Soft Computing, 2023: 2029064. https://doi.org/10.1155/2023/2029064

[4] Niu, W.N., Wang, Y.H., Liu, X.Y., Yan, R., Li, X., Zhang, X. (2023). GCDroid: Android malware detection based on graph compression with reachability relationship extraction for IoT devices. IEEE Internet of Things Journal, 10(13): 11343-11356. https://doi.org/10.1109/JIOT.2023.3241697

[5] Mobile Operating System Market Share Worldwide|Statcounter Global Stats—gs.statcounter.com. https://gs.statcounter.com/os-market-share/mobile/worldwide, accessed on Aug. 22, 2023.

[6] Oh, T., Stackpole, B., Cummins, E., Gonzalez, C., Ramachandran, R., Lim, S. (2012). Best security practices for android, blackberry, and iOS. In 2012 The First IEEE Workshop on Enabling Technologies for Smartphone and Internet of Things (ETSIoT), Seoul, pp. 42-47. https://doi.org/10.1109/ETSIOT.2012.6311252

[7] Mobile Cyberthreat Report for 2022—securelist.com. https://securelist.com/mobile-threat-report-2022/108844/, accessed on Aug. 22, 2023.

[8] Ren, Y.J., Leng, Y., Cheng, Y.P., Wang, J. (2019). Secure data storage based on blockchain and coding in edge computing. Mathematical Biosciences and Engineering, 16(4): 1874-1892. https://doi.org/10.3934/mbe.2019091

[9] Detecting and Eliminating Chamois, a Fraud Botnet on Android—Android-developers.googleblog.com.

https://android-developers.googleblog.com/2017/03/detecting-and-eliminating-chamois-fraud.html, accessed on Aug. 22, 2023.
[10] Malware Statistics & Trends Report|AV-TEST—av-test.org. https://www.av-test.org/en/statistics/malware/, accessed on Aug. 22, 2023.
[11] Kouliaridis, V., Kambourakis, G. (2021). A comprehensive survey on machine learning techniques for android malware detection. Information, 12(5): 185. https://doi.org/10.3390/info12050185
[12] Chen, Y.C., Chen, H.Y., Takahashi, T., Sun, B., Lin, T. (2021). Impact of code deobfuscation and feature interaction in Android malware detection. IEEE Access, 9: 123208-123219. https://doi.org/10.1109/ACCESS.2021.3110408
[13] Alabrah, A. (2023). A novel neural network architecture using automated correlated feature layer to detect Android malware applications. Mathematics, 11(20): 4242. https://doi.org/10.3390/math11204242
[14] Tarwireyi, P., Terzoli, A., Adigun, M.O. (2022). BarkDroid: Android malware detection using bark frequency cepstral coefficients. Indonesian Journal of Information Systems, 5(1): 48-63. https://doi.org/10.24002/ijis.v5i1.6266
[15] Fan, M., Liu, J., Luo, X.P., Chen, K., Tian, Z.Z., Zheng, Q.H., Liu, T. (2018). Android malware familial classification and representative sample selection via frequent subgraph analysis. IEEE Transactions on Information Forensics and Security, 13(8): 1890-1905. https://doi.org/10.1109/TIFS.2018.2806891
[16] Fatima, A., Kumar, S., Dutta, M.K. (2020). Host-server-based malware detection system for android platforms using machine learning. In Advances in Computational Intelligence and Communication Technology, vol 1086. Springer, Singapore, pp. 195-205. https://doi.org/10.1007/978-981-15-1275-9_17
[17] Cai, H.P., Jenkins, J. (2018). Poster: Towards sustainable android malware detection. In 2018 IEEE/ACM 40th International Conference on Software Engineering: Companion (ICSE-Companion), Gothenburg, Sweden, pp. 350-351. https://doi.org/10.1145/3183440.3195004
[18] Fang, Z., Liu, J., Huang, R.B., Chen, P., Li, X., Chen, X. (2021). Research on multi-model android malicious application detection based on feature fusion. In 2021 4th International Conference on Robotics, Control and Automation Engineering (RCAE), Wuhan, China, pp. 147-151. https://doi.org/10.1109/RCAE53607.2021.9638928
[19] Surendran, R., Thomas, T., Emmanuel, S. (2020). A TAN based hybrid model for android malware detection. Journal of Information Security and Applications, 54(3): 102483. https://doi.org/10.1016/j.jisa.2020.102483
[20] Al Ali, M., Svetinovic, D., Aung, Z., Lukman, S. (2018). Malware detection in Android mobile platform using machine learning algorithms. In 2017 International Conference on Infocom Technologies and Unmanned Systems (Trends and Future Directions) (ICTUS), Dubai, United Arab Emirates. https://doi.org/10.1109/ICTUS.2017.8286109
[21] Cai, H.P., Meng, N., Ryder, B., Yao, D. (2019). DroidCat: Effective Android malware detection and categorization via app-level profiling. In IEEE Transactions on Information Forensics and Security, 14(6): 1455-1470. https://doi.org/10.1109/TIFS.2018.2879302
[22] Cao, K.L., Welham, Z.M. (2021). Principal component analysis (PCA). Multivariate Data Integration Using R, Chapman and Hall/CRC. https://doi.org/10.1201/9781003026860-12
[23] Uddin, M.P., Mamun, M.A., Afjal, M.I., Hossain, M.A. (2020). Information-theoretic feature selection with segmentation-based folded principal component analysis (PCA) for hyperspectral image classification. International Journal of Remote Sensing, 42(1): 286-321. https://doi.org/10.1080/01431161.2020.1807650
[24] Alsadi, A.A., Sameshima, K., Bleier, J., Yoshioka, K., Lindorfer, M., Eeten, M.V., Gañán, C.H. (2022). No spring chicken: Quantifying the lifespan of exploits in IoT malware using static and dynamic analysis. In Proceedings of the 2022 ACM on Asia Conference on Computer and Communications Security, 309-321. https://doi.org/10.1145/3488932.3517408
[25] Costa, F.H., Medeiros, I., Menezes, T., Silva, J., Silva, I.L., Bonif'acio, R., Narasimhan, K., Ribeiro, M. (2021). Exploring the use of static and dynamic analysis to improve the performance of the mining sandbox approach for android malware identification. Journal of Systems and Software, 183: 111092. https://doi.org/10.1016/j.jss.2021.111092
[26] Gera, T., Singh, J., Faruki, P., Thakur, D. (2022). Efficacy of Android security mechanisms on ransomware analysis and detection. AIP Conference Proceedings, 2357(1): 040007. https://doi.org/10.1063/5.0080931
[27] Amer, E., Mohamed, A. (2022). Using machine learning to identify android malware relying on API calling sequences and permissions. Journal of Computing and Communication, 1(1): 38-47. https://doi.org/10.21608/jocc.2022.218454
[28] Ashmore, R., Calinescu, R., Paterson, C. (2021). Assuring the machine learning lifecycle: Desiderata, methods, and challenges. ACM Computing Surveys, 54(5): 1-39. https://doi.org/10.1145/3453444
[29] Pasdar, A., Lee, Y.C., Hong, S.H. (2023). Catch the Intruder: Collaborative and Personalized malware detection by on-device application fingerprinting. In 2023 IEEE International Conference on Web Services (ICWS), Chicago, IL, USA, pp. 595-604. https://doi.org/10.1109/ICWS60048.2023.00078
[30] Razgallah, A., Khoury, R., Hallé, S., Khanmohammadi, K. (2021). A survey of malware detection in Android apps: Recommendations and perspectives for future research. Computer Science Review, 39: 100358. https://doi.org/10.1016/j.cosrev.2020.100358
[31] Sihag, V., Vardhan, M., Singh, P. (2021). A survey of android application and malware hardening. Computer Science Review, 39: 100365. https://doi.org/10.1016/j.cosrev.2021.100365
[32] Wang, S.S., Yan, Q.B., Chen, Z.X., Yang, B., Zhao, C., Conti, M. (2017). Detecting Android malware leveraging text semantics of network flows. IEEE Transactions on Information Forensics and Security, 13(5): 1096-1109. https://doi.org/10.1109/TIFS.2017.2771228
[33] Zhu, H.J., You, Z.H., Zhu, Z.X., Shi, W.L., Chen, X., Cheng, L. (2018). DroidDet: Effective and robust detection of android malware using static analysis along with rotation forest model. Neurocomputing, 272: 638-646. https://doi.org/10.1016/j.neucom.2017.07.030

[34] Google Play Python API. (2021) https://github.com/fahadakbar24/google-play-api, accessed on Aug. 22, 2023.
[35] Malware Dataset. (2021) https://github.com/fahadakbar24/android-malware-detection-dataset, accessed on Aug. 22, 2023.
[36] Androguard: Reverse Engineering, Malware Analysis of Android Applications. (2021) Available online: https://github.com/androguard/androguard, accessed on Aug. 22, 2023.