Grenze International Journal of Engineering and Technology, June Issue

Malware Analysis for Android-based Reverse Engineering Techniques

Aniruddh Dewri[1], Omar Shaikh[2] and Nilima Dongre[3]
[1-3]Ramrao Adik Institute of Technology, Department of IT, D Y Patil Vidyapeeth, Navi Mumbai, India
Email: aniruddh.dewri@gmail.com, omarshaikh19989@gmail.com, nilima.dongre@rait.ac.in

***Abstract*— Mobile devices are one of the most popular and widespread technological implements in the modern world. Their immense popularity represents a proportionate immense vulnerability to mobile device-based malware attacks, especially on Android phones, which have cornered the majority of the market. Hackers are constantly devising new ways of attacking smartphones using creative and clever techniques like surveillance, credential theft, and malicious advertising. Therefore, there arises a need to safeguard against such attacks and secure our devices. One promising avenue is the use of machine learning (ML)-based methods to develop specific classifiers based on a set of training examples, which can be relied on by malware detectors instead of using definitions for each malware signature. In this paper, we review the ML techniques for malware analysis of Android apps. We devise our own ML method based on suitability and offer a justification. We also delve into reverse engineering of Android applications, and determine the overall security and vulnerability of a few chosen datasets. In doing so, we hope to demonstrate any issues in this field, which can be the subject of further research.**

***Index Terms*— Malwre Analysis, Android, Reverse Engineering**

I. INTRODUCTION

In this technological age, the use of smartphones and related applications is quickly rising as a result of the ease and functionality of various applications and the constant development of smart devices and software. The number of smartphone users is expected to reach 4.3 billion by 2023. The most popular mobile operating system (OS) is Android. As of May 2021, its market share is 72.2%. The second highest market share belongs to Apple iOS with 26.99%. The remaining 0.81% is shared by Samsung, KaiOS and other smaller companies.

As of May 2021, the number of applications published exceeded 2.9 million. Among them, more than 2.5 million applications are classified as normal applications, while 400,000 applications are classified as bad applications by AppBrain. Android's global reputation makes it an attractive target for cybercriminals and more vulnerable to malware and viruses. Research suggests many ways to detect these attacks, and machine learning is one of the most important [4]. This is because machine learning methods can infer classifications from training examples. Therefore, using samples eliminates the need to specify signatures when building malware detectors. Interpreting signatures requires a combination of skill and hard work. In some cases, although there are no clear rules (signatures), examples can be obtained. A lot of work and research has been done on machine learning-centred malware detection on Android, the main focus of this article. Android users and developers are known to make

Grenze ID: 01.GIJET.10.2.567

mistakes that expose them to unnecessary risks and risks to their devices containing malware. Therefore, the malware detection process as well as the way to identify these errors are important and will be explained in this article.

Using machine learning to detect malware involves two main steps: analyzing the Android application package (APK) to obtain the appropriate system, and machine and deep learning (DL) methods based on the results to identify malicious APKs. Therefore, existing methods for APK analysis are reviewed, including static, dynamic and hybrid analyse. As in malware detection, finding software code vulnerabilities involves two primary steps: generating features through code analysis and machine learning of the resulting features to identify negative numbers. Therefore, both were included in the classification analysis.

II. Literature Review

Ying et al. in [1] propose a new method for imbalanced learning, called AWGSENN, which uses adaptive weighting and Gaussian function synthesizing to generate synthetic minority class samples. AWGSENN is shown to outperform other resampling methods, such as ADASYN and SMOTE, on a dataset of Android malware detection. However, the authors note that AWGSENN is unsuitable for datasets where no neighbour belongs to the majority class in the ENN data clearing step.

Aqil et al. in [2] propose a method for Android malware detection based on network traffic using the decision tree algorithm. The method involves feature analysis and extraction, data cleaning, feature selection, and model training and testing. The Extra Trees classifier achieved the highest accuracy in malware detection, and conversation-based features enhanced the results. The authors suggest that adding static features could further enhance the detection of malware and its categorization.

Samaneh et al. in [3] propose a semi-supervised deep learning approach for Android malware category classification. The approach involves a 5-step process: data collection, analysis, pre-processing, training, and detection. The approach is shown to be effective even with a small number of labelled training samples. However, the authors note that their dataset is unreliable, with only 13077 out of 17341 APKs running successfully in the CopperDroid analysis tool.

Yujin et al. in [4] propose a technique called ModelAttacker to generate adversarial attacks against deep learning models on Android apps. ModelAttacker takes a deep learning model as input and derives a similar model. It then trains the parameters of adversarial attacks on the derived model. The authors show that targeted adversarial attacks are significantly more successful than blind attacks. However, they also acknowledge that ModelAttacker may fail to derive a similar model to the input model, which could limit its effectiveness.

Gianni et al. in [5] propose a method for Android malware family classification using dynamic features and neural networks. Feature extraction and analysis, dynamic behaviour representation, and classification utilising recurrent and convolutional neural networks (RNNs) are all part of the process. The authors demonstrate how, when the appropriate features are used, neural networks can effectively identify different virus families. However, they also note that the proposed approach demands a large number of features, and the dataset used in their evaluation saw 33% of inputs be unusable.

Faris in [6] provides a systematic review of Android malware detection techniques, classifying them into static, dynamic, and hybrid techniques. It highlights the trade-offs involved in choosing between different techniques, and it can help researchers and developers to make informed decisions about which technique to use in a given context.

Yue et al. in [7] investigate the performance of machine learning (ML)-based Android malware detection models in the presence of temporal inconsistency in the evaluation dataset. The authors find that the detection performance of these models is significantly improved if temporal inconsistency is introduced. Further analysis reveals that ML models learn to distinguish malware from benign apps based on temporal differences, rather than the actual malicious behaviours. This results in extremely poor performance when the models are evaluated on datasets with different temporal distributions than the training data.

Umm-el-Hani et al. in [8] survey recent trends in deep learning-based malware detection. The authors highlight the potential of meta learning-based algorithms for producing generic models that can be trained for self-learning to detect unseen malware types. However, they also note that new forms of malware keep emerging, and future research should focus on developing generic models that can detect zero-day malware.

Jaykumar et al. in [9] survey the use of explainable graph neural networks (GNNs) for cyber malware analysis. The authors emphasize the bright future of graph machine learning techniques used in virus detection., as GNNs

can capture the complex relationships between different components of malware. However, they also note that a large and diversified dataset would be needed to effectively compare the metrics of different models.
Pascal et al. in [10] examine current developments in deep learning models for mobile and desktop malware detection. In their discussion of deep learning models for malware detection, the authors cover activation strategies, regularisation, loss functions, and network optimizers. They also talk about several features that are extracted for malware detection and methods for malware analysis. The majority of deep learning malware detection models now in use, according to the authors, are unsuitable for Android mobile devices and do not operate in real-time. This is due to the fact that deep learning models may be memory-intensive and computationally costly.
Farnood et al. in [11] propose a new Android malware detection method called AIM, which uses a neural network classifier and hybrid analysis to differentiate malicious software from legitimate programmes. Using a revolutionary approach to class modelling, AIM also detects harmful classes within applications. In comparison to other approaches of a similar nature, the suggested class modelling technique improves malware detection accuracy and allows AIM to identify harmful code snippets within programs.
Leonardo et al. in [12] propose a lightweight, multi-stage method for Android malware detection that uses non-invasive machine learning algorithms called Malware APK Detection Solution (MADS). Compared to other cutting-edge techniques, MADS has been demonstrated to be three times faster and use ten times less CPU resources. Nevertheless, the study did not examine how changes in the confidence level values' threshold impact the categorization outcomes. This is an important consideration, as the threshold value can significantly impact the accuracy and precision of the malware detection model.
Ismail et al. in [13] propose a new ensemble approach for Android malware detection based on fuzzy logic and machine learning classifiers. The proposed model is called FL-BDE, and it combines the outputs of six different machine learning classifiers using fuzzy logic. FL-BDE is compared with other state-of-the-art ensemble-based malware detection models, and it is shown to perform better. The authors also suggest that hybrid models, which combine static and dynamic analysis techniques, may perform even better.
Ye et al. in [14] propose a novel data pre-processing attack technique that can be used to inject malicious code into deep learning (DL) models in Android apps. The attack is successful against 81% of the DL apps tested by the authors. However, some apps cannot be installed after the malicious code is injected. The attack works by targeting the data pre-processing step of the DL model. This step is responsible for preparing the data for input to the model. The attackers can inject malicious code into the data pre-processing step by reverse engineering the DL app and modifying the code. The malicious code can be used to perform a variety of attacks, such as stealing user data, damaging the device, or even taking control of the device.

III. Proposed System

In this section, we define the steps taken to detect and classify Android apps as malware or benign, as shown in Figure 1.

A. Reverse Engineering

The first step is reverse engineering the Android application package (APK) file. For this, we implemented a frontend web application built with HTML and CSS, to provide a user-friendly interface for uploading. The backend is a Node.js server with Express middleware, which handles the upload request and script execution. Here is where we upload the APK file to be analysed. The file is then sent to the server, which utilises a Python script executing APKtool, a useful and versatile tool for performing reverse engineering, on the target file. The reverse engineered APK file is stored as an XML file (AndroidManifest.xml)

B. Feature Extraction

The first step in our data transformation process is the extraction of permissions from the APK files. We use the xml.etree ElementTree module to parse the AndroidManifest.xml file, which contains the permissions used by the application. The extract_permissions function is designed for this purpose. It takes the path of the AndroidManifest.xml file as input and returns a list of permissions used by the application.

C. Data Transformation

Once the permissions are extracted, we transform this data into a format suitable for our machine learning models. We create a DataFrame df_input where each column corresponds to a permission and each row corresponds to an application. The value in each cell indicates whether the application uses the permission (1) or not (0). This binary representation is a simple yet effective way to encode categorical data for machine learning models. However, our

machine learning models are trained on a specific set of permissions (columns in the training data). To ensure compatibility between our input data and the models, we reindex df_input to have the same columns as our training data. Any missing columns in df_input are filled with zeros, indicating that the corresponding permissions are not used by the application. The data is then cast to integer type to ensure computational efficiency.

Figure 1: Flowchart of proposed system

D. Training the Classification Models

We employ machine learning models based on five different algorithms: Naïve Bays, decision tree, K Nearest Neighbours (KNN), Support Vector Machine (SVM), and Random Forest. Each ML model has its own set of hyperparameters that need to be tuned to achieve the best performance. Some of the most common hyperparameters for these models include:

- SVM: The kernel type, C parameter, and gamma parameter.
- RF: The number of trees, the maximum depth of each tree, and the number of features to consider at each split.
- DT: The maximum depth of the tree, the minimum number of samples in a node, and the minimum number of samples required to split a node.
- KNN: The number of neighbours to consider, the distance metric used to calculate distances between points, and the weights assigned to different neighbours.
- Naive Bayes: The smoothing parameter used to prevent overfitting.

Each model is trained and tested on a general .csv dataset containing a large number of extracted permissions. The trained models are then used for detection.

E. Detection

Each classification model takes the input data and classifies the APK as malware or benign. As well, the corresponding classification measures are calculated and published as output.

IV. Performance Analysis

In this section, we evaluate the effectiveness and efficiency of the machine learning-based framework in classifying Android APK files into one of two pre-defined categories: Malware or Benign. This evaluation is crucial in understanding the practicality of our approach in real-world applications.

A. Environment

The proposed machine learning-based framework was implemented on a PC equipped with a 3.60 GHz Core i7-4790 CPU and 32 GB RAM. This setup ensures that we have sufficient computational resources to handle the demands of training multiple machine learning models and processing large datasets. Our implementation was done in Python, a popular language in the data science community due to its simplicity and the availability of numerous scientific computing libraries. We leveraged several of these libraries in our implementation, including pandas for data manipulation, sklearn for machine learning, and matplotlib for data visualization.

The machine learning models used in this study include Naive Bayes, Decision Tree, K-Nearest Neighbors, SVM, and Random Forest. Each model was chosen for its unique strengths in handling classification tasks. The input layer of these models consists of the number of permissions used by the APK files, serving as the feature set for the models. The output layer consists of two neurons, corresponding to the two categories of APK files (Malware or Benign). For models that benefit from hyper parameter tuning, such as K-Nearest Neighbors, SVM, and Random Forest, we employed GridSearchCV. This method performs an exhaustive search over specified parameter values for an estimator, optimizing the parameters for model performance.

B. Dataset

The dataset used in this study is a CSV file loaded using the pandas library. This file contains information about various APK files, including their permissions and whether they are classified as Malware or Benign. The dataset was split into a training set and a test set using a test size of 0.20, ensuring that the models have unseen data to test on after training. The 'type' column of the dataset, which indicates whether the APK is Malware or Benign, was used as the target variable for the models. This is the variable that our models will attempt to predict based on the permissions used by the APK files. To gather the permissions used by each APK file, the APK file was reverse-engineered using APKtool. This tool decompiles the APK file and extracts its contents, including the AndroidManifest.xml file. This file was then parsed to extract the permissions it uses. These permissions were then used as input for the machine learning models.

C. Experimental Results

In our experiments, we used a variety of metrics to assess the overall classification performance of our machine learning models. These metrics included precision (PR), recall (RC), F1-Score (F1), accuracy (ACC), false positive rate (FPR), false negative rate (FNR), and classification error. A false positive in our context is a benign APK being detected as malicious, and a false negative is a malicious APK being detected as benign. We employed several machine learning models in our experiments, including Naive Bayes, Decision Tree, K-Nearest Neighbors (KNN), Support Vector Machine (SVM), and Random Forest. For the KNN, SVM, and Random Forest models, we used GridSearchCV to find the best hyperparameters. The models were trained on a dataset loaded from a CSV file, with a train-test split of 80-20. The dataset was composed of various features extracted from APK files, with the 'type' column indicating whether the APK is benign or malicious.

After training, each model's performance was evaluated on the test set, and the results were displayed in terms of accuracy and a classification report. Additionally, the actual and predicted values were plotted for visual comparison. For the APK files, we extracted permissions from the AndroidManifest.xml file using the xml.etree.ElementTree module. The extracted permissions were then transformed into a binary format suitable for our machine learning models. Any missing permissions in the input data were filled with zeros. Finally, the trained models were used to predict the class (benign or malicious) of new APK files. The prediction results were then written to an HTML file, along with the classification graphs for each model.

Table 1 displays a comparison of the performance metrics of the various ML models we used for classification.

As shown in the table, the decision tree and Support Vector Machine models achieved the best performances on the dataset, with accuracies of 95% and 94%, respectively. The Naïve Bayes, K-nearest neighbours and Random Forest models also achieved good performance, with accuracies of 85%, 84%, and 86%, respectively.

Figures 2, 3, 4, 5 and 6 display five graphs representing the accuracy of the five models, Naïve Bays, K-nearest neighbours, decision trees, Support Vector Machines (SVM) and Random Forest (RF), respectively, when trained on and applied to our dataset.

Figure 7 shows a comparison the results of the five classification models on the dataset. The comparison results of analysis and classification techniques of related research works are presented in Table 2. It displays the classification techniques used, the dataset employed as well as a few results metrics such as accuracy, precision, recall, and F1-score

TABLE I: DATASETS AND EVALUATION RESULTS OF PROPOSED SYSTEM

Dataset	Techniques used	Accuracy	Precision	Recall	F1-score
Mahindru. Arvind (2018), "Android Permission dataset, V1, doi	Naïve Bays	0.83	0.85	0.84	0.84
	KNN	0.95	0.95	0.95	0.95
	Decision tree	0.85	0.87	0.85	0.85
	SVM	0.95	0.95	0.95	0.95
	Random forest	0.86	0.87	0.86	0.86

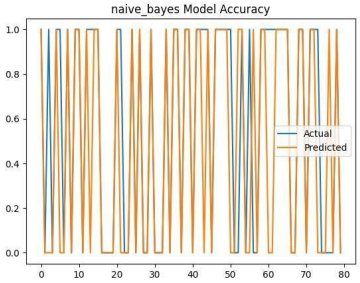

Figure 2: Accuracy of Naive Bayes model

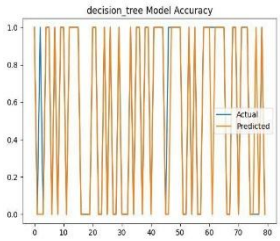

Figure 3: Accuracy of Decision Tree model

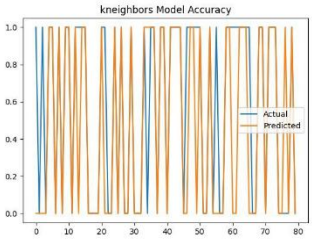

Figure 4: Accuracy of K Neighbors model

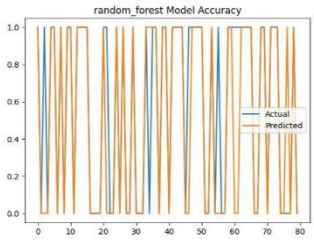

Figure 5: Accuracy of Random Forest model

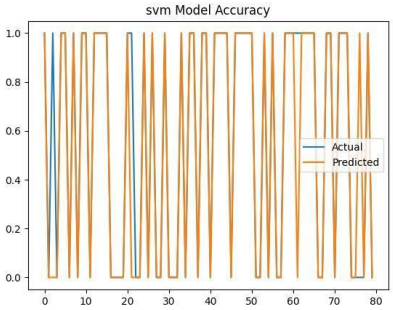

Figure 6: Accuracy of SVM model

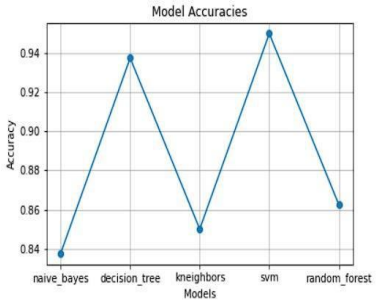

Figure 7: Comparison results of the 5 models

Table III shows a comparison of results obtained in research papers and literature on similar topics, acquired by conducting literature review.

TABLE II: REAL-TIME PERFORMANCE METRICS ON A SINGLE APK

	Precision	Recall	F1-score	Support
Accuracy	1.0	1.0	1.0	1.0
Macro average	1.0	1.0	1.0	1.0
Weighted average	1.0	1.0	1.0	1.0

TABLE III: DATASETS AND EVALUATION RESULTS OF LITERATURE EMPLOYING CLASSIFICATION ALGORITHMS

References	Techniques used	Dataset	Accuracy	Precision	Recall	F1-score
Aqil et al., 2019, [2]	Decision tree	Drebin dataset, Contagion dumpset	0.98, 0.97	-	-	-
Samaneh et al., 2020, [3]	PLDNN	CICMalDroid2020	0.96	0.99	0.96	0.97
Yujin et al., 2021, [4]	RNN, CNN	Airpush, Dowgin, FakeInst, DroidKungFu (D.K. Fu), Opfake	0.99, 0.99	0.99, 0.99	-	0.99, 0.99
Leonardo et al. 2023, [12]	Random Forest	CIC-AndMal2017 CICMalDroid 2020 and R- PackDroid	0.99	-	-	-
Ismail Atacak 2023, [13]	SVM, RF	Drebin dataset	0.94, 0.96	0.94, 0.97	0.94, 0.95	0.94, 0.96

V. CONCLUSION

In this research paper, we develop an Android malware detection system using reverse engineering, static analysis, and classification algorithms. Our technology is capable of extracting many static properties from APK archives, including permissions, API calls, and image streaming controls. We then use various classification algorithms such as K Nearest Neighbors, Decision Trees and Naive Bayes to predict whether an APK is malware or not. Our test results show that our technology can detect malware. The closest K model and SVM model achieved the best performance with 95% accuracy. Decision trees and Random Forest models also performed well, with 85% and 86% accuracy, respectively.

Our technology has many advantages over other malware detection methods. First, it can detect malware without scanning time. This makes it more efficient and less resource intensive than dynamic analysis techniques. Second, our technology can detect new and evolving malware, even if they have never been seen before. This is because our system is based on static functionality that does not change much from one malware variant to another.

Overall, our results show that our malware detection tool is a good way to protect Android users from malicious applications. However, further research is needed to evaluate the performance of our method on larger datasets and to develop the countermeasures against malware strains.

REFERENCES

[1] Pang, Ying, et al. "Imbalanced learning based on adaptive weighting and Gaussian function synthesizing with an application on Android malware detection." Information Sciences 484 (2019): 95-112.

[2] Zulkifli, Aqil, et al. "Android malware detection based on network traffic using decision tree algorithm." Recent Advances on Soft Computing and Data Mining: Proceedings of the Third International Conference on Soft Computing and Data Mining (SCDM 2018), Johor, Malaysia, February 06-07, 2018. Springer International Publishing, 2018.

[3] Mahdavifar, Samaneh, et al. “Dynamic android malware category classification using semi-supervised deep learning.” 2020 IEEE Intl Conf on Dependable, Autonomic and Secure Computing, Intl Conf on Pervasive Intelligence and Computing, Intl Conf on Cloud and Big Data Computing, Intl Conf on Cyber Science and Technology Congress (DASC/PiCom/CBDCom/CyberSciTech). IEEE, 2020.

[4] Huang, Yujin, Han Hu, and Chunyang Chen. “Robustness of on-device models: Adversarial attack to deep learning models on android apps.” 2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering in Practice (ICSESEIP). IEEE, 2021.

[5] D’Angelo, Gianni, et al. “Effective classification of android malware families through dynamic features and neural networks.” Connection Science 33.3 (2021): 786-801.

[6] Alharbi, Faris Auid, Abdurhman Mansour Alghamdi, and Ahmed S. Alghamdi. “A Systematic Review of Android Malware Detection Techniques.” International Journal of Computer Science and Security (IJCSS) 15.1 (2021): 1-19. 26

[7] Liu, Yue, et al. “Explainable AI for android malware detection: Towards understanding why the models perform so well?.” 2022 IEEE 33rd International Symposium on Software Reliability Engineering (ISSRE). IEEE

[8] Tayyab, Umm-e-Hani, et al. “A survey of the recent trends in deep learning-based malware detection.” Journal of Cybersecurity and Privacy 2.4 (2022): 800-829.

[9] Warmsley, Dana, et al. “A survey of explainable graph neural networks for cyber malware analysis.” 2022 IEEE International Conference on Big Data (Big Data). IEEE, 2022.

[10] Maniriho, Pascal, Abdun Naser Mahmood, and Mohammad Jabed Morshed Chowdhury. “A Survey of Recent Advances in Deep Learning Models for Detecting Malware in Desktop and Mobile Platforms.” arXiv preprint arXiv:2209.03622 (2022).

[11] Faghihi, Farnood, Mohammad Zulkernine, and Steven Ding. “AIM: An Android Interpretable Malware detector based on application class modeling.” Journal of Information Security and Applications 75
[12] Da Costa, Leonardo, and Vitor Moia. “A Lightweight and Multi-stage Approach for Android Malware Detection using Non-invasive Machine Learning Techniques.” IEEE Access (2023).
[13] Atacak, ˙Ismail. “An Ensemble Approach Based on Fuzzy Logic Using Machine Learning Classifiers for Android Malware Detection.” Applied Sciences 13.3 (2023): 1484.
[14] Sang, Ye, et al. “Beyond the Model: Data Pre-processing Attack to Deep Learning Models in Android Apps.” arXiv preprint arXiv:2305.03963 (2023)