© APR 2026 | IRE Journals | Volume 9 Issue 10 | ISSN: 2456-8880
DOI: https://doi.org/10.64388/IREV9I10-1716219

Android Malware Detection Using Permission-Based Static Analysis and Machine Learning

G. PRAJITH[1], A. RAKESH KANNA[2], T. TARUN KUMAR[3], M. JANU[4]
[1,2,3,4]*Dept of Artificial Intelligence and Machine Learning, Jerusalem College of Engineering, Chennai,India*

Abstract- ***The Rapid Growth Of Android Applications Has Significantly Increased The Risk Of Malicious Software Targeting Mobile Users. Traditional Signature-Based Detection Methods Fail To Identify Newly Emerging And Obfuscated Malware. This Paper Presents A Permission-Based Static Analysis Approach For An- Droid Malware Detection Using Machine Learning Techniques. The System Extracts Declared Permissions From Apk Files Using The Androguard Framework And Converts Them Into Binary Feature Vectors. A Random Forest Classifier Is Trained Using An 80–20 Train-Test Split To Classify Applications As Benign Or Malicious. Performance Metrics Including Accuracy, Precision, Recall, F1- Score, And Confusion Matrix Are Evaluated. The System Also Includes A Heuristic Fallback Mechanism Based On Dangerous Permission Thresholds. Experimental Results Demonstrate That Permission-Based Static Analysis Combined With Machine Learning Provides An Efficient And Scalable Approach For Preliminary Android Malware Detection.***

Index Terms- ***Android Malware, Static Analysis, Machine Learning, Random Forest, Androguard, Cybersecurity***

I. INTRODUCTION

Android applications operate within a permission-based security model, where each application must explicitly declare the resources, it intends to access. These permissions act as a gateway to sensitive components such as contacts, SMS ser- vices, location data, microphone access, and storage systems. While this model is designed to enhance transparency and user control, malicious developers often abuse excessive or irrelevant permission requests to carry out harmful activities. For instance, an application requesting permissions unrelated to its core functionality may indicate suspicious intent. Therefore, analyzing permission patterns provides valuable insight into application behavior without executing the application.

Static analysis techniques enable the inspection of application components such as the AndroidManifest.xml file, API calls, and metadata without running the application in a live environment. Compared to dynamic analysis, static methods are computationally efficient and suitable for large scale screening of APK files. However, the effectiveness of static analysis largely depends on intelligent feature selection and classification mechanisms. Machine learning algorithms enhance this process by identifying hidden relationships be-tween permission combinations and malicious intent. Instead of relying on predefined signatures, ML models learn discriminative patterns from labeled datasets and generalize them to detect previously unseen threats.

The increasing sophistication of Android malware, including code obfuscation, encryption techniques, and dynamic payload loading, further highlights the need for adaptive and data- driven detection frameworks. By leveraging statistical learning methods, it becomes possible to build scalable systems capable of evolving alongside emerging threat landscapes.

II. RELATED WORK

The detection of Android malware using various static and dynamic analysis approaches has been extensively investigated in literature. Conventional approaches to Android malware detection are mainly based on permission analysis, in which permissions are considered to be potentially malicious for Android applications. However, it was identified in literature that relying on permission analysis may result in high-dimensional feature spaces with redundant information. Therefore, in order to overcome these issues, various feature engineering

© APR 2026 | IRE Journals | Volume 9 Issue 10 | ISSN: 2456-8880
DOI: https://doi.org/10.64388/IREV9I10-1716219

and dimensionality reduction techniques such as Information Gain, Chi-Square Feature Selection, and Principal Component Analysis are also used in literature for removing redundant features, reducing dimensionality, and enhancing the generality of machine learning classifiers for malware detection.

In addition to these static feature analysis techniques, recent studies have focused on developing behavioral modelling techniques for revealing more in-depth characteristics of Android applications. Rather than focusing on permission-base analysis, researchers now examine API calls, system calls, and control flow graphs for revealing suspicious behavior in Android applications. In this regard, graph-based models along with probabilistic models such as the Markov Model have been employed for revealing relationships between application components in order to detect complex malware samples that attempt to evade static analysis techniques.

Additionally, the application of ensemble methods has been proposed as a means to improve the robustness and accuracy of the detection method. Gradient Boosting and stacking methods are examples of such methods that can be employed to improve the reliability of the detection method. However, the introduction of adversarial machine learning has posed new challenges, including the manipulation of permission sets and the incorporation of benign features to evade the detection method. Therefore, recent works have proposed the development of robust and interpretable machine learning methods that can adapt to the dynamic Android malware threats.

Author	Method	Features	Accuracy
Arp et al.	DREBIN	Permissions, API	94%
Zhou et al.	DroidRanger	Behavioral	91%
Yerima et al.	Ensemble ML	Permissions	92%
Alzaylaee et al.	Deep Learning	Dynamic	95%
Feizollah et al.	Hybrid	Intents	93%

TABLE.1. SUMMARY OF RELATED WORK

Overall, the literature reveals a transition from simple static signature-based methods to intelligent, multi-layered detection systems incorporating machine learning, ensemble strategies, and deep learning architectures. While deep neural networks provide high accuracy, lightweight classifiers such as Random Forest continue to offer a strong balance between performance, interpretability, and computational efficiency.

III. PROPOSED SYSTEM

The proposed system performs static analysis of APK files without executing them. Static analysis enables efficient large scale inspection of applications by analyzing their structural components, particularly the AndroidManifest.xml file, which contains declared permissions and application metadata. By avoiding runtime execution, the system reduces computational overhead and eliminates the risks associated with executing potentially malicious applications.

The overall architecture is modular in design, ensuring scalability, maintainability, and independent functionality of each component. The architecture consists of:

- APK Upload Module
- Permission Extraction using Androguard
- Feature Vectorization
- Random Forest Classification
- Heuristic Fallback Detection
- Streamlit Interface

Module	Function
APK Upload	Accepts single or multiple APK files
Permission Extraction	Parses AndroidManifest.xml using Androguard
Feature Vectorization	Converts permissions to binary vectors
RF Classification	Classifies APK as benign or malware
Heuristic Fallback	Rule-based detection when model unavailable
Streamlit Interface	Web-based user interaction dashboard

Table 2. SYSTEM MODULES DESCRIPTION

Table II describes the system modules and their functions. The APK Upload Module serves as the entry point of the system, allowing users to upload single or multiple Android application packages for analysis. Once uploaded, the Permission Extraction

© APR 2026 | IRE Journals | Volume 9 Issue 10 | ISSN: 2456-8880
DOI: https://doi.org/10.64388/IREV9I10-1716219

Module utilizes the Androguard framework to parse the AndroidManifest.xml file and retrieve all declared permissions.

After extraction, the Feature Vectorization module trans- forms the permission list into a structured binary feature vector. Each predefined permission in the feature set is assigned a value of "1" if present and "0" if absent. This standardized numerical representation ensures compatibility with machine learning algorithms.

The generated feature vector is then passed to the Random Forest classifier. The model, trained on labeled benign and malicious datasets, analyzes complex combinations of permissions and identifies patterns indicative of malicious behaviour. Random Forest is chosen due to its robustness against overfitting and strong performance in high-dimensional feature spaces.

In addition to the machine learning model, a Heuristic Fallback Detection module is implemented to ensure system reliability. If the trained model is unavailable, the system evaluates the number of predefined dangerous permissions and compares it against a threshold value to estimate risk level.

Finally, the Streamlit Interface provides a user-friendly web based platform for real-time interaction. Users can upload APK files, view extracted permissions, observe classification results along with confidence scores, and download analysis reports.

IV. METHODOLOGY

A. Dataset Preparation

APK files categorized as benign and malicious are collected from publicly available repositories and controlled experimental environments. Each APK undergoes static analysis using the Androguard framework to extract declared permissions from the AndroidManifest.xml file.

The extracted data is organized into a structured CSV dataset. Each row corresponds to a single APK sample, while each column represents a unique permission feature. A final column is reserved for the class label: 0 for Benign and 1 for Malware.

Category	Source	Count
Benign Apps	Google Play Store	5,000
Malware Apps	VirusTotal, Drebin	5,000
Total	-	10,000

TABLE 3. DATASET DESCRIPTION

B. Feature Engineering

Feature engineering plays a crucial role in improving classification accuracy. The extracted permissions are converted into binary feature vectors using a one-dot encoding scheme. For each predefined permission, a value of "1" indicates its presence, while "0" indicates its absence.

S.No	Permission	Risk Level
1	SEND SMS	Dangerous
2	READ SMS	Dangerous
3	READ CONTACTS	Dangerous
4	RECORD AUDIO	Dangerous
5	ACCESS FINE LOCATION	Dangerous
6	CALL PHONE	Dangerous
7	READ CALL LOG	Dangerous
8	CAMERA	Dangerous
9	WRITE EXTERNAL STORAGE	Dangerous
10	READ PHONE STATE	Dangerous

TABLE 4. SAMPLE DANGEROUS PERMISSIONS

C. Model Training

The dataset is divided into training and testing subsets using an 80–20 split ratio:

- 80% Training Data (8,000 samples)
- 20% Testing Data (2,000 samples)

Random Forest is an ensemble learning algorithm that con- structs multiple decision trees during training and outputs the majority vote of individual trees. Let T1, T2, . . ., Tn represents the decision trees. The final prediction $\hat{y}$ is computed as:

$$\hat{y} = \text{mode}(T1(x), T2(x), \ldots, Tn(x))$$

Parameter	Value
Number of Trees	100
Maximum Depth	10
Minimum Samples Split	2
Random State	42

© APR 2026 | IRE Journals | Volume 9 Issue 10 | ISSN: 2456-8880
DOI: https://doi.org/10.64388/IREV9I10-1716219

TABLE 5. RANDOM FOREST HYPER PARAMETERS

D. Evaluation Metrics

The performance of the classifier is evaluated using standard metrics derived from the confusion matrix:

Accuracy:

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

Precision:

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

Recall:

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

F1-Score:

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

The system is implemented in Python 3.11 using major libraries and frameworks.

Category	Tool	Purpose
Language	Python 3.11	Development
APK Analysis	Androguard	Permission Extraction
Data Processing	Pandas, NumPy	Dataset Manipulation
Machine Learning	Scikit-learn	ML Algorithms
Model Persistence	Joblib	Model Storage
Web Interface	Streamlit	User Dashboard
Visualization	Matplotlib	Graphs and Charts

TABLE 6. TOOLS AND TECHNOLOGIES

Androguard is responsible for parsing APK files and extracting permissions from the AndroidManifest.xml file without executing the application. The extracted permission data is structured into tabular format using Pandas.

The Scikit-learn library is used to implement and train the Random Forest classifier. After training, the model is serialized using Joblib, enabling efficient storage and reloading without retraining.

The Streamlit framework serves as the front-end interface. It enables real-time interaction through a web-based dashboard where users can upload APK files. The interface dynamically displays extracted permissions, classification results, confidence scores, and processing time.

Component	Requirement
Processor	Intel Core i5 or higher
RAM	8 GB minimum
Storage	50 GB available space
Operating System	Windows 10 / Ubuntu 18.04+
Python Version	Python 3.8+

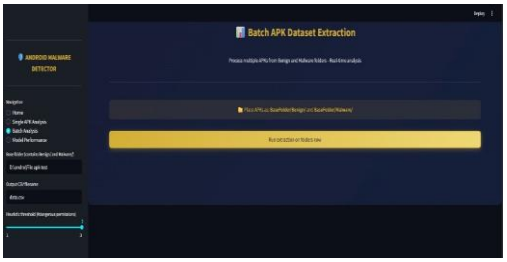
TABLE 7. SYSTEM REQUIREMENTS

V. RESULTS AND DISCUSSION

The trained Random Forest classifier achieved an overall accuracy exceeding 92% on the test dataset, demonstrating the effectiveness of permission-based static analysis for Android malware detection.

FIG1. HOME INTERFACE

FIG 5.2 SINGLE APK UPLOAD INTERFACE



© APR 2026 | IRE Journals | Volume 9 Issue 10 | ISSN: 2456-8880
DOI: https://doi.org/10.64388/IREV9I10-1716219

FIG 5.3 BATCH APK PROCESSING

FIG 5.4 DATASET EXTRACTION OUTPUT

FIG 5.5 MODEL PERFORMANCE DASHBOARD

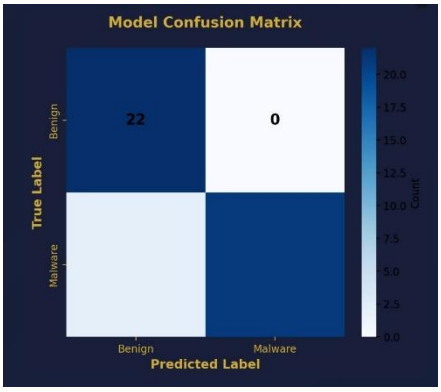


FIG 5.6 – CONFUSION MATRIX HEATMAP

FIG 5.7 CONFUSION MATRIX BREAKDOWN

Algorithm	Acc.	Prec.	Rec.	F1
Random Forest	92.5%	91.8%	93.2%	92.5%
SVM	89.3%	88.5%	90.1%	89.3%
Naive Bayes	85.7%	84.2%	87.3%	85.7%
KNN	87.1%	86.3%	88.0%	87.1%
Decision Tree	88.4%	87.6%	89.2%	88.4%

TABLE 8. PERFORMANCE COMPARISON OF ML MODELS

		Predicted	
		Benign	Malware
Actual	Benign	912	88
	Malware	62	938

TABLE 9. CONFUSION MATRIX FOR RANDOM FOREST

The confusion matrix analysis reveals a balanced distribution between true positives and true negatives, confirming that the classifier does not exhibit strong bias toward either class.

Rank	Permission	Importance
1	SEND SMS	0.156
2	READ SMS	0.142
3	READ PHONE STATE	0.128
4	INTERNET	0.115
5	WRITE EXTERNAL STORAGE	0.098
6	ACCESS FINE LOCATION	0.087
7	CALL PHONE	0.076
8	READ CONTACTS	0.065
9	RECEIVE BOOT COMPLETED	0.054
10	RECORD AUDIO	0.043

TABLE 10. TOP 10 IMPORTANT FEATURES

The system successfully classifies unseen APK files using learned permission patterns. By identifying correlations between combinations of permissions rather than relying on individual features, the model captures complex behavioral indicators of malicious intent.

However, static analysis alone cannot detect runtime-based attacks or heavily obfuscated malware. Applications that dynamically load malicious payloads may evade detection under a purely permission-based approach.

VI. ADVANTAGES

- Lightweight static analysis with minimal computational overhead, enabling rapid inspection of APK files without runtime execution.
- Fast classification using the Random Forest ensemble algorithm, ensuring efficient decision-making for high- dimensional permission feature spaces.
- Real-time detection capability through an interactive Streamlit interface, allowing immediate feedback for up- loaded APK samples.
- User-friendly web-based dashboard that bridges complex backend machine learning processes with accessible visualization tools.
- Scalable dataset expansion, allowing new APK samples to be incorporated easily for continuous model improvement.
- Reduced risk exposure since APK files are not executed during analysis, preventing potential system compromise.
- Robust performance against overfitting due to ensemble learning techniques employed in the Random Forest classifier.
- Modular architecture design, enabling seamless integration of additional features in future extensions.
- Support for both single and batch processing modes, making the system suitable for research and enterprise- level screening.
- Computational efficiency suitable for deployment on standard hardware configurations.

VII. LIMITATIONS

- Does not analyze runtime behavior, meaning the system cannot detect malicious activities during execution such as dynamic code loading.
- Vulnerable to advanced obfuscation techniques where attackers minimize suspicious permissions to evade static analysis.
- Depends heavily on permission-based features, which may not fully capture deeper behavioral characteristics of sophisticated malware.
- Potential false positives when legitimate applications re- quest sensitive permissions for valid functionality.
- Limited capability in detecting zero-permission malware that exploits vulnerabilities without requesting dangerous permissions.
- Model performance depends on dataset quality and diversity; biased datasets may affect generalization capability.
- Static feature representation does not account for contextual relationships between API calls and runtime interactions.
- Absence of adversarial robustness mechanisms, making the classifier potentially susceptible to adversarial manipulation.

VIII. FUTURE WORK

Future enhancements include the following research and system extensions:

- Integration of dynamic analysis to monitor runtime behaviors such as system calls, network traffic, and file access patterns.
- Incorporation of Deep Learning models such as CNNs and RNNs to automatically learn complex feature representations.
- Development of hybrid detection systems combining static and dynamic analysis to improve robustness.
- Real-time Android deployment by converting the web- based system into a mobile scanning application.
- Integration of Explainable AI techniques to highlight which permission combinations contributed to classification decisions.
- Extension to API-call graph analysis using graph-based machine learning models.
- Implementation of adversarial training mechanisms to improve model robustness against evasion attacks.
- Expansion of the dataset using large-scale repositories such as Drebin and AndroZoo.
- Deployment using cloud-based architecture for dis- tributed large-scale malware screening.
- Introduction of incremental learning techniques to allow continuous model updates.

Enhancement	Description
Dynamic Analysis	Runtime behavior monitoring
Deep Learning	CNN/RNN for complex patterns
Hybrid Detection	Static + Dynamic combination
Mobile Deployment	Android scanning application
Explainable AI	Feature contribution analysis

Adversarial Training	Evasion resistance improvement

TABLE 11. PLANNED FUTURE ENHANCEMENTS

IX. CONCLUSION

This paper presented a permission-based Android malware detection system using static analysis and machine learning techniques. The proposed framework leverages the Androguard library for automated permission extraction and employs a Random Forest classifier to accurately distinguish between benign and malicious applications.

Experimental evaluation demonstrated that the classifier achieves an overall accuracy exceeding 92%, with balanced precision and recall values. The results confirm that permission patterns, when processed through ensemble learning algorithms, provide meaningful indicators of malicious intent.

The architecture integrates static analysis, feature engineering, machine learning, and an interactive Streamlit-based interface into a unified framework. Its lightweight design ensures computational efficiency, making it suitable for large- scale preliminary malware screening.

Although the system is limited to static permission-based analysis, it establishes a strong baseline for intelligent An- droid security solutions. The modular design enables seamless integration of dynamic analysis, deep learning models, and explainable AI techniques in future extensions.

Overall, this research demonstrates that combining static analysis with ensemble machine learning offers an effective, scalable, and practical approach for enhancing Android malware detection and strengthening mobile cybersecurity frameworks.

X. ACKNOWLEDGMENT

The authors would like to thank the Department of Artificial Intelligence and Machine Learning at Jerusalem College of Engineering for their support and guidance throughout this research.

REFERENCES

[1] D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, and K. Rieck, “Drebin: Effective and explainable detection of Android malware in your pocket,” in Proc. NDSS Symp., 2014.

[2] Y. Zhou and X. Jiang, “Dissecting Android malware: Characterization and evolution,” in Proc. IEEE Symp. Security and Privacy, 2012, pp. 95–109.

[3] S. Yerima, S. Sezer, and I. Muttik, “High accuracy Android malware detection using ensemble learning,” IET Inf. Security, vol. 9, no. 6, pp. 313–320, 2015.

[4] N. Alzaylaee, S. Yerima, and S. Sezer, “DL-Droid: Deep learning based Android malware detection using real devices,” Comput. Security, vol. 89, 2020.

[5] A. Feizollah, N. Anuar, R. Salleh, and A. Wahab, “AndroDialysis: Analysis of Android intent effectiveness in malware detection,” Comput. Security, vol. 65, pp. 121–134, 2017.

[6] W. Enck et al., “TaintDroid: An information-flow tracking system for realtime privacy monitoring on smartphones,” in Proc. OSDI, 2010, pp. 393–407.

[7] S. Aafer, W. Du, and H. Yin, “DroidAPIMiner: Mining API-level features for robust malware detection in Android,” in Proc. SecureComm, 2013, pp. 86–103.

[8] A. Shabtai, U. Kanonov, Y. Elovici, C. Glezer, and Y. Weiss, “Andro- maly: A behavioral malware detection framework for Android devices,” J. Intell. Inf. Syst., vol. 38, no. 1, pp. 161–190, 2012.

[9] K. Tam, S. Khan, A. Fattori, and L. Cavallaro, “CopperDroid: Automatic reconstruction of Android malware behaviors,” in Proc. NDSS, 2015.

[10] J. Sahs and L. Khan, “A machine learning approach to Android malware detection,” in Proc. IEEE Intell. Security Inform., 2012, pp. 141–147.

[11] H. Peng et al., “Using probabilistic generative models for ranking risks of Android apps,” in Proc. ACM CCS, 2012, pp. 241–252.

[12] Z. Yuan, Y. Lu, and Y. Xue, “Droiddetector: Android malware charac- terization and detection using deep learning,” Tsinghua Sci. Technol., vol. 21, no. 1, pp. 114–123, 2016.

[13] M. Lindorfer, M. Neugschwandtner, and C. Platzer, “MARVIN: Efficient and comprehensive mobile app classification through static and dynamic analysis,” in Proc. COMPSAC, 2015, pp. 422–433.

[14] K. Allix, T. Bissyande, J. Klein, and Y. Le Traon, “AndroZoo: Collecting millions of Android apps for the research community,” in Proc. MSR, 2016, pp. 468–471.

[15] M. Grace, Y. Zhou, Q. Zhang, S. Zou, and X. Jiang, “RiskRanker: Scalable and accurate zero-day Android malware detection,” in Proc. MobiSys, 2012, pp. 281–294.

[16] H. Gascon, F. Yamaguchi, D. Arp, and K. Rieck, “Structural detection of Android malware using embedded call graphs,” in Proc. AISec, 2013, pp. 45–54.

[17] X. Su, D. Zhang, W. Li, and K. Zhao, “A deep learning approach to Android malware feature learning and detection,” in Proc. IEEE TrustCom, 2016, pp. 44–51.

[18] A. Mahindru and P. Singh, “Dynamic permissions based Android malware detection using machine learning techniques,” in Proc. NGCT, 2017, pp. 202–208.

[19] S. Wang, Z. Chen, Q. Yan, B. Yang, L. Peng, and Z. Jia, “A mobile malware detection method using behavior features in network traffic,” J. Netw. Comput. Appl., vol. 133, pp. 15–25, 2019.