IJIRAE:: International Journal of Innovative Research in Advanced Engineering ISSN: 2349-2163
Volume 13, Issue 01, January 2026 https://www.ijirae.com/archives
https://doi.org/10.26562/ijirae.2026.v1301.02

Android Malware Detection

Prof.K.Shilpa Reddy
Assistant Professor, Department of Computer Science and Engineering,
Vemana Institute of Technology Bengaluru, India
shilpa.reddy@vemanait.edu.in
https://orcid.org/0000-0002-2970-1279
Abhispoorthi Y V, B G Deeksha Srie, Balam Vasavi, Charita Y
Department of Computer Science and Engineering,
Vemana Institute of Technology Bengaluru, India
abhispoorthi29@gmail.com, ddeeksha741@gmail.com
vasavireddy628@gmail.com, charithay2004@gmail.com

Publication History
Manuscript Reference No: IJIRAE/RS/Vol.13/Issue01/AEJA26.JAAE10081
Research Article | Open Access | Double-Blind Peer-Reviewed | Article ID: IJIRAE/RS/Vol.13/Issue01/AEJA26.JAAE10081
Received:12, December 2025, Revised: 24, December 2025, Accepted: 02, January 2026, Published Online:20, January 2026.
https://www.ijirae.com/volumes/Vol13/iss-01/02.AEJA26.JAAE10081.pdf
Article Citation:Shilpa,Abhispoorthi,Deeksha,Balam,Charita(2026),Android Malware Detection, IJIRAE: International Journal of Innovative Research in Advanced Engineering,Volume 13, Issue 01 of 2026 pages 06-11
Doi:>https://doi.org/10.26562/ijirae.2026.v1301.02
BibTeX Key: Shilpa@2026Android
IJIRAE papers should be cited as IJIRAE (International Journal of Innovative Research in Advanced Engineering, AM Publications, India 2025, ISSN 2349-2163, https://doi.org/10.26562/ijirae.2026.v1301.02 The journal's official abbreviation is IJIRAE. Orcid: https://orcid.org/0009-0004-9398-7488

Abstract: Machine learning based detection system of Android malware and analysis of features which are static. The deigned system extracts non executable features such as permissions, intents, activities, and API calls from Android APK files. We analyze non-executable APK features and classify them using a Random Forest model deployed on a Flask server. Our Kotlin application interacts with this backend via REST APIs for real-time prediction. The Android application developed in Kotlin communicates with the backend using REST APIs to perform malware prediction in real time. Training of the model was done using datasets from Debrin, AndroZoo, and custom APK collections, accuracy of 95 Percentage, precision of 94 percentages, and recall of 93 percentages was achieved. The proposed solution provides an effective, scalable, and lightweight detection of malware approach, enhancing mobile security without relying on dynamic or runtime analysis.

Keywords: Android Security, Malware Detection, Static Analysis, Machine Learning, Random Forest, Flask API, Mobile Security.

I. INTRODUCTION

Android is the world's more usable mobile operating system, making it a prime target for malware attacks. Because Android dominates the global smart phone market, attackers frequently exploit its openness, making it the main focus of mobile malware campaigns. Most traditional antivirus solutions depend either on dynamic execution or are resource-intensive, or cannot identify new and obfuscated malwares, which has also been mentioned in various sys call- based detection researches, one of that is research done by Surendran et al.[1]. In this scenario, a static feature based on techniques in machine learning was presented. The system extracts metadata directly from APK files, instead of executing them, hence enabling fast, safe, and lightweight analysis apt for mobile devices. This corresponds to previous static and hybrid feature-based research, such as Kadir and Peddoju [2]. From the proposed system's perspective in a three-tier architecture, including the Android front end, Flask backend, and machine learning model, real-time malware predictions can be delivered. Beyond this basic level, the approach utilizes permissions, activities, services, intents, and other manifest-level attributes in order to create feature vectors, classifying them with a previously trained Random Forest model. This is in conformance with the effectiveness of ML-based classifiers discussed in Kushwahaetal.[3]. Due to performing all the Computation intensive tasks in a cloud backend, this not only requires fewer resources on the device but also enables continuous updates of models without any change on the mobile application side. It will ensure that scalability, adaptability to newly emerging malware families, and enhanced detection performance compared to traditional signature-based techniques are offered. Overall, it offers a practical, effective, and user-friendly solution for enhancing Android security through combining the strengths of static feature analysis and machine learning.

II. PROBLEM STATEMENT

With the rapid proliferation of Android devices, malware targeting these platforms has significantly increased in both volume and sophistication. The majority of mobile security applications rely on signature-based detection, which struggles against zero-day malware or obfuscated malicious apps. Storing and updating large malware signature databases in mobile phones also consumes memory and processing power. It is necessary to have a scalable, intelligent, and efficient malware detection system.

III. LITERATURE SURVEY

Roopak et al. [1] developed a detection method which relies on patterns of system calls to identify malicious behavior. Their technique converts the presence of syscall sequences into binary features, making the system fast and suitable for low end devices. However, because the approach does not analyze syscall arguments, it lacks deeper behavioral context. As a result, attackers could modify syscall sequences to resemble legitimate application behavior, which might reduce the method's effectiveness against mimicry-based malware.

Sateesh and Peddoju [2] proposed a hybrid a malware detection method that integrates multiple features static characteristics of Android apps, including permissions, API usage, opcodes, and component information. By merging these diverse features, their model achieved improved accuracy and better adaptability when measured on the Omni Droid dataset. Using standard scaling also enhanced the machine learning algorithms performance such as KNN and LR. While the model is computationally efficient, it only focuses on static analysis and does not capture behaviors that occur during execution. This limitation can make it less effective against obfuscated or dynamically triggered malware. The authors also emphasize the importance of evaluating their method on larger and more varied datasets to make the system more reliable in real-world environments.

Kushwaha et al. [3] implemented a malware detection approach using a Multi-Layer Perceptron (MLP) classifier, which demonstrated strong performance on both known and slightly modified malicious applications. Since the model learns generalized behavior patterns rather than relying on exact signatures, it can detect previously unseen threats more effectively than signature-based tools. The authors compared the MLP with other models of machine learning across multiple datasets to evaluate its effectiveness. However, they observed that the accuracy can decrease when the dataset size and diversity increase, highlighting the model's dependence on the training data. Therefore, extensive evaluation across use of datasets is important to ensure stable performance in real-world use cases.

Joomye, Jasser, and Meehongling [4] introduced a method of detection of malware which uses Temporal Convolutional Networks (TCNs) to extract dynamic behavioral features during app execution. TCNs are capable of handling long sequence data more efficiently and with greater stability than recurrent neural networks such as RNNs or LSTMs, so that they become suitable for analysis complex run time behavior found in advanced malware. Their experiments showed that the TCN-based model achieved better accuracy and recall than many learning techniques. However, the study's dataset included a limited number of benign samples, leading to class imbalance and a higher risk of false positive results. Additionally, the approach only distinguishes between benign and malicious apps and does not identify specific malware families, reducing its usefulness for detailed threat analysis or digital forensics.

Gupta, Sharma, and Garg [5] proposed a lightweight hybrid detection technique that combines selected static properties with behavioral features to identify malicious applications. This "feature-pair bonding" strategy improves classification accuracy while keeping the low computational cost, enabling it to run on low-resource devices. However, it becomes difficult to manually generating and pairing the features while handling large datasets or performing real-time analysis. Additionally, the approach lacks interpretability since it fails to clearly show specific feature pairs contribute to the final classification outcome. The system's effectiveness also depends strongly on the relevance and quality of the extracted features. Rui Shen et al. [6] explored a transfer learning-based approach where Android Dex files are turned into image representations, allowing deep learning models to identify malware through visual patterns. By using pre-trained models and then fine-tuning them on malware datasets, this method reduces the need for large Android-specific labeled data. The results showed strong performance, especially against highly obfuscated malware, since structural similarities can still be recognized in image form. However, converting applications into images and processing them through deep networks introduces significant computational cost, making the approach unsuitable for real-time detection on mobile devices. Additionally, the system must be frequently updated to maintain accuracy as new malware strains emerge.

Jinsung Kim et al.[7] presented a approach that employs deep learning on dynamically collected behavioral data from Android applications, executed either on-device or in the cloud. Since the system monitors actual runtime activity instead of relying only on static information, it offers strong accuracy and can detect sophisticated threats that disguise their malicious intent. However, the approach needs power of high computation because complex model processing and Dex-to-image conversion, making it challenging to deploy efficiently on mobile devices. Updating model frequently is also necessary to stay effective against newly emerging malware variants. Even with cloud assistance, the overall system remains heavy for seamless, low-latency deployment on smart phones. Basha et al. [8] introduced a detection strategy that uses Genetic Algorithms for selecting the most relevant static features before applying machine learning classification. This optimization reduces feature dimensionality, making training faster while preserving strong detection accuracy, including for some unknown threats. However, discarding weaker but potentially valuable features may slightly affect precision. Additionally, because it relies only on static properties of the application, malicious actions that activate only during execution may go undetected, limiting the system's effectiveness against advanced runtime-based attacks.

IV. METHODOLOGY

The newly proposed method for detecting Android malware has been designed using a structured multi-stage methodology that integrates dataset preparation, feature extraction, machine-learning model development, backend API creation, and deployment of the Android application. Our pipeline enables real-time malware detection by combining cloud processing with static analysis.

A. Dataset Collection

To build an effective and reliable malware detection system, the first step is gathering a dataset that contains a wide variety of legitimate and malicious Android applications. In this study, samples were sourced from two well-known research repositories: Drebin and AndroZoo. Drebin offers labeled malware belonging to numerous threat families, making it suitable for training the model to differentiate between multiple types of harmful behaviors. In contrast, AndroZoo provides huge APKs, including many benign apps, which helps the dataset reflect realistic usage patterns found in the ecosystem of android. In total, 15,036 APK files were selected for experimentation, consisting of 9,476 benign and 5,560 malicious applications. This distribution ensures the dataset remains representative while avoiding excessive class imbalance that could bias the model. Each APK was analyzed using the Androguard framework to extract meaningful static attributes from the manifest and application structure. These include permissions requested by the app, declared components such as activities and services, intent filters, API call information and certain metadata such as SDK version and file size. All retrieved data was transformed into a structured 216- dimensional numerical feature vector after preprocessing. This standardized feature format enables machine learning classifiers to learn patterns that distinguish malware from normal applications. By using multiple feature categories and diverse data sources, the proposed system aims to improve its capability so that both known and evolving Android threats can be detected.

Table I. Dataset Summary

Attribute	Details
Dataset Sources	Drebin + Andro Zoo
Total Samples	15,036
BenignAPKs	9,476
Malicious APKs	5,560
Feature Dimensions	216
Feature Type	Static (Manifest-based)

Table I presents the distribution of labels, confirming a realistic dataset: there is a higher number of benign samples, corresponding to the actual behavior of mobile app ecosystems. This balanced composition of the dataset allows for better generalization of the classifier to known and unknown malware threats.

B. Static Feature Extraction

Andro guard framework is used to collect meaningful static features from each APK. These include permissions, activities, services, intents, SDK version, file size, and other manifest level attributes. Numerical representations of these features are extracted and used for machine learning. The reason for choosing analysis of static is that is safe, fast, and avoids executing potentially harmful applications.

C. Model Training

These preprocessed features serve as input for a machine- learning classifier. Several models were compared, but the Random Forest/XGBoost classifier provided the best trade- off between accuracy, precision, and low false-positive rate. We split the dataset into 80:20, training and testing, respectively, for proper model validation. This model be serialized using job lib to be integrated into the backend server for use.

D. Backend API Development

ARESTful API built with Flask enables easy communication between the Android application and the machine-learning model. APKs uploaded through the application get their processing at the server itself, where the model is deployed, classifying them and returning results to the mobile client. The API takes care of feature extraction, model inference, and response generation. Therefore, in this client–server architecture, computation is handled in the cloud rather than overloading user devices.

E. Android Application Development

The developed Android application is fully functional, using Kotlin, which allows users to register, login, and upload APKs directly from their smart phones. The uploaded APKs are passed to the backend server through REST API calls, where the results pertaining to detection-benign or malicious- are shown to the user with confidence scores. It provides a very simple, fast, and user-friendly interface for real-time malware detection.

F. Deployment

The final stage is deploying the model of machine learning and backend server to a cloud platform. The infrastructure must be such that real-time access, scalability, and continuous updating of the model can be supported. The Android app is integrated with the cloud server such that end users can remotely perform malware detection.

V. SYSTEM ARCHITECTURE

The system follows the client–server architecture where malware detection is carried out on the cloud so that it minimizes processing overhead at the user's device. The architecture consists of four major components:

A. Android Application

The mobile application, developed in Kotlin, enables users to select and upload APK files. The communication of the app with the backend server is done securely through RESTAPIs and shows the final classification result as benign or malicious along with the confidence score. This provides an easy-to-use interface for real-time malware scanning.

IJIRAE:: International Journal of Innovative Research in Advanced Engineering ISSN: 2349-2163
Volume 13, Issue 01, January 2026 https://www.ijirae.com/archives
https://doi.org/10.26562/ijirae.2026.v1301.02

B. Cloud Server

The cloud server functions as the core processing unit. It receives the uploaded APK file, forwards it for feature extraction, and then passes the extracted features to the machine-learning model. The server returns the prediction to the Android application after classification. Cloud off loading ensures scalability and enables continuous model updates without requiring changes in the user app. Fig 1 describes the 3w3proposed system is cloud-based, providing a client–server architecture in which the uploaded APK file by the Android application is forwarded to the cloud server for analysis. The server sends the APK to the feature- extraction module for static analysis, and the result ant feature vector feeds the machine-learning classifier that classifies the app as either benign or malicious. The final prediction, in addition to malware probability score, gets relayed to the Android application. In this way, the computation complexity is shifted to the cloud, allowing the mobile devices to conduct swift and lightweight malware detection.

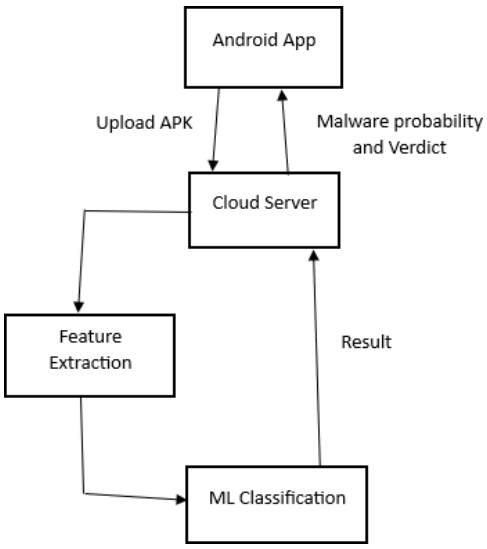

Fig 1: System Architecture of Android Malware Detection

C. Feature Extraction Module

This module carries out static analysis without executing the APK. It uses the Androguard tool for feature extraction such as permissions, activities, services, intent, SDK information, and other metadata. These are then converted into numerical feature vectors suitable for machine-learning algorithms

D. Machine Learning Classification

A pre-trained ML model, such as Random Forest or XG Boost, then takes the extracted feature vector to identify the application as malicious or benign. The classifier also generates a confidence score. Hosting the model in cloud makes updates efficient, thus making the solution more adaptable to emerging malware threats.

VI. DATAFLOW

It also represents how the user interacts with the Android application, the cloud server, dataset, and the machine learning model. The system works in two major steps that you can see in fig 2.

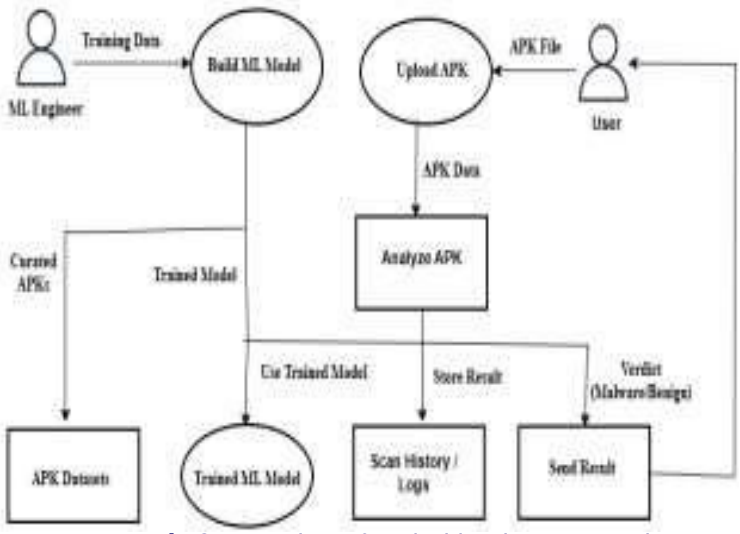

Fig 2: Data Flow of Android Malware Detection

It consists of an offline model development phase and an online detection phase. During the offline phase, a dataset comprising benign and malicious APKs is collected. The features like permissions, activities, services, and intents are then extracted for training machine-learning classifier. The developed model will be deployed on a cloud server for real- time inference. At the online phase, the user can upload an APK from an Android application that securely forwards the uploaded file to the cloud for static analysis. The obtained features will be fed into the stored model for classifying applications between benign and malicious, along with the calculation of the confidence score. The prediction will be logged for future reference, and the final result will immediately be presented to the user on the mobile interface.

VII. RESULTS AND ANALYSIS

To asses show well the proposed Android Malware Detection System performs, the random forest classifier was trained on a dataset collected from Drebin and AndroZoo, supplemented with additional custom APKs. Then, the model was evaluated using unseen samples in order to validate its generalization capability. In the confusion matrix presented in Fig.3, which says model correctly predicted 1885 benign and 1085 malicious applications. The misclassifications were only 11 for benign and 27 for malicious apps, thus demonstrating a very small error rate and strong capability for threat detection. Table II: Performance Evaluation Metrics further confirms the robustness through the detailed classification report. By this the system was able to achieve 99 % precision, 98 % recall, and 99 % F1- score, thus providing an overall accuracy of 98.7%. The high recall rate regarding malicious samples signifies that even challenging malware variants can be detected by this model with fewer false negatives, which is essential in security applications. The integrated Android client with the cloud-based backend provides for fast and efficient scanning and could deliver malware predictions within a few seconds of APK upload. This architecture greatly reduces the device resource consumption while ensuring high scalability and support for continuous model updates. These results confirmed that the proposed approach can provide a very accurate, lightweight, and practical solution for real-time Android malware detection.

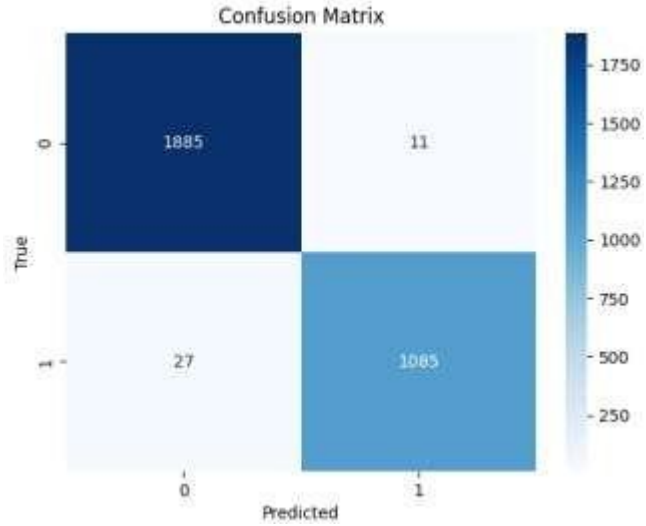

Fig 3: Confusion Matrix of the Random Forest classifier for Android Malware Detection.

Table II. PERFORMANCE EVALUATION METRICS OF THE PROPOSED MODEL

Metric	Benign (Class0)	Malicious (Class 1)	Overall Score
Precision	0.99	0.99	0.99
Recall	0.99	0.98	0.99
F1-Score	0.99	0.98	0.99
Accuracy	—	—	0.987

VIII. CONCLUSION AND FUTURE WORK

The system designed was able to effectively embeding machine learning in the cloud for real-time malware classification. Using static features extracted from Android APKs and a Random Forest classifier deployed on a Flask backend, the system achieves highly reliable performance with 98.7% accuracy, 99% precision, and 98% recall. A lightweight Android application communicates with the server through secure REST APIs and delivers fast predictions without consuming device resources. It follows that the efficacy of the system in detecting both known malware and unseen variants makes it a valid candidate to improve mobile security in a scalable and user-friendly way. The system demonstrates good performance, but further improvement of malware detection capabilities can still be achieved. Some of the future improvements may include dynamic analysis for better detect obfuscated and evolving threats. Generalization is best enhanced through the expansion of datasets with more malware families sourced from large-scale repositories such as Virus Share, supplemented with real-time threat intelligence feeds. Approaches to be considered in order to improve feature representation include deep learning and hybrid models.

On-device acceleration or federated learning could be used for system deployment to reduce network dependency and improve privacy. These developments will lead to an ever- improving level of protection against emerging Android malware.

ACKNOWLEDGMENT

The authors sincerely thank their project guide for the continuous guidance, motivation, and valuable feedback throughout the development of this work. The authors also acknowledge the Department of Computer Science and Engineering, Vemana Institute of Technology, Bengaluru, for providing the required resources and support to successfully complete this project. The collaborative effort of all team members contributed significantly to the implementation this Android Malware Detection System successfully.

REFERENCES

1. Roopa k surendran, MD Meraj Uddin, Tony Thomas, and Gokul Pradeep, “Android Malware Detection Based on Informative Syscall Subsequences”,2024,1-11. https://doi.org/10.1109/ACCESS.2024.3387475
2. Abdul Kadir, Sateesh K.Peddoju, “Android Malware Detection using Hybrid Features and Machine Learning”,2024, 1-2. https://doi.org/10.1109/MASS62177.2024.00077
3. Pradeep Kumar Kushwaha, Vikas Kumar, Manish Saraswat, PankajSingh,“Android Malware Detection and its Security”,2023,1-5.
4. Abdurraheem Joomye, Muhammed basheer Jasser, Meehongling, “An Effective Temporal Convolutional Networks-Based Method for Detecting Android Malware Using Dynamic Extracted Features”,2025,1-14.
5. Rahul Gupta, Kapil Sharma, R.K. Garg, “Android Malware Detection based on Feature pair Bonding: A Hybrid Detection Model”,2023,1-4.
6. Rui Shen, Hui-juan Zhu, Chang Li, Hua-hui Wei, “GAResNet: A Transfer Learning based Framework for Android Malware Detection”,2023,1-6. https://doi.org/10.1109/ICKG59574.2023.00038
7. Jinsung Kim, Younghoon Ban, Eunbyeol Ko, Haehyun Cho, Jeong Hyun Yi, “MAPAS: a practical deep learning- based android malware detection system”,2022, 1-14.
8. V Basha,Gorre Ashmitha Reddy, Cheruku Sadhana, Balthu Manideep, Kolukula Laxman, “Android Malware Detection Using Genetic Algorithm based on optimized feature selection and Machine Learning”,1-9.