International Journal of Engineering Research and Science & Technology

www.ijerst.org

ISSN : 2319-5991

Vol. 22 No. 2 (2026)



ijerst.editor@gmail.com

editor@ijerst.com

International Journal of Engineering Research and Science & Technology

ISSN 2319-5991 www.ijerst.com
Vol. 22, No. 2, 2026
© 2026 IJERST. All Rights Reserved
Peer Reviewed, Referred & Indexed Journal

ANALYSIS AND DETECTION OF MALWARE IN ANDROID APPLICATIONS USING MACHINE LEARNING

1.Y.SureshBabu Assot.prof CSE dept,Gokula Krishna College of Engineering,Sullurpet,Tirupati Dist ,AP

2.T.Bhagyasri, T.Deepthi, M.Subhash, A.Charan, R.Moula,CSE dept,Gokula Krishna College of Engineering,Sullurpet,Tirupati District,AP

ABSTRACT

The rapid expansion of Android applications has significantly increased the risk of mobile malware, necessitating advanced and efficient detection mechanisms. This study presents a hybrid machine learning-based framework for Android malware detection that integrates both static and dynamic analysis to improve classification accuracy and reliability. Static features such as permissions and intent filters are extracted through reverse engineering of application files, while dynamic features, including API call sequences, are captured during runtime execution in a controlled environment. Unlike conventional approaches that rely on single-feature analysis, the proposed system combines these heterogeneous features to capture both structural and behavioral characteristics of applications.

To enhance detection performance, the extracted features are preprocessed and transformed into structured representations suitable for machine learning models. A Support Vector Machine (SVM) classifier is employed as the primary detection model due to its robustness in handling high-dimensional data and classification boundaries. Experimental observations indicate that dynamic features provide better detection capability compared to static features, while the hybrid approach achieves superior overall performance. The proposed method demonstrates improved accuracy, reduced false positives, and better generalization, making it suitable for real-time Android malware detection and mobile security applications.

Keywords— Android security, malware detection, hybrid feature analysis, machine learning classification, support vector machine, static and dynamic analysis, API behavior modeling, permission-based analysis, mobile threat detection, anomaly detection, cybersecurity analytics.

I. INTRODUCTION

The rapid advancement of mobile technologies has led to an exponential increase in the usage of Android-based applications across various domains such as communication, banking, healthcare, and entertainment. Due to its open-source nature and widespread adoption, the Android operating system has become a prime target for cyber threats, particularly malware attacks. Malware, which refers to malicious software designed to compromise system integrity, can lead to unauthorized access, data theft, financial loss, and privacy breaches. As mobile devices increasingly store sensitive personal and organizational data, ensuring their security has become a critical research concern [1].

Traditional malware detection techniques, such as signature-based approaches, are no longer sufficient to handle modern threats. These methods rely on predefined patterns and fail to detect new or evolving malware variants, especially those employing obfuscation, polymorphism, and code transformation techniques [2]. Consequently, there is a growing need for intelligent and adaptive detection mechanisms capable of identifying unknown threats in real time.

Machine learning-based approaches have emerged as a promising solution for malware detection. These approaches analyze application features and learn patterns that distinguish malicious applications from benign ones. Feature extraction plays a crucial role in this process, as the quality of extracted features directly impacts detection performance [3]. Android malware detection typically relies on two major analysis techniques: static analysis and dynamic analysis.

Static analysis involves examining application code without execution, extracting features such as permissions, intent filters, and API usage. Although it is computationally efficient, static analysis is vulnerable to code obfuscation and may fail to capture runtime behavior [4], [5]. On the other

hand, dynamic analysis observes application behavior during execution in a controlled environment, capturing features such as API call sequences, system calls, and network activities. While dynamic analysis provides more accurate behavioral insights, it is resource-intensive and time-consuming [6], [7].

To overcome the limitations of individual techniques, hybrid analysis combines both static and dynamic features, offering improved detection accuracy and robustness. Recent studies have shown that integrating multiple feature types enhances the capability of detecting sophisticated malware variants [8], [9].

In addition, efficient feature representation techniques such as hashing and tokenization have been introduced to handle large-scale datasets and high-dimensional feature spaces. These methods reduce computational complexity while preserving important information required for classification [10]. Machine learning classifiers such as Support Vector Machine (SVM), Random Forest (RF), Logistic Regression (LR), and Naïve Bayes (NB) are widely used for malware detection due to their ability to handle complex data patterns [11]–[13].

This work focuses on developing a hybrid malware detection system that leverages both static and dynamic features and applies machine learning techniques for classification. The proposed approach aims to improve detection accuracy, reduce false positives, and enhance scalability for real-world deployment.

II. LITERATURE SURVEY

Several research efforts have been made to address the challenges of Android malware detection using different analysis techniques and machine learning models. Early approaches primarily relied on static analysis, where application features such as permissions and API calls were extracted from the code without execution. Studies have shown that permission-based detection methods can achieve reasonable accuracy; however, they often suffer from high false positive rates due to overlapping permissions between benign and malicious applications [14], [15].

To improve detection performance, researchers have explored combining permissions with other static features such as intent filters and API calls. These approaches demonstrated enhanced accuracy compared to single-feature models by capturing more contextual information about application behavior [16]. However, static analysis techniques remain ineffective against advanced malware that employs code obfuscation and encryption strategies to hide malicious intent [17].

Dynamic analysis has been introduced to overcome these limitations by analyzing application behavior during runtime. This approach captures detailed execution traces, including API call sequences, system interactions, and network activities. Studies utilizing API call sequences as features have reported high detection accuracy, as these features provide a comprehensive view of application behavior [18]. Nevertheless, dynamic analysis introduces additional computational overhead and requires specialized environments such as sandboxes or virtual machines, limiting its scalability [19].

Hybrid analysis methods have gained significant attention due to their ability to combine the strengths of both static and dynamic approaches. By integrating structural and behavioral features, hybrid models achieve higher accuracy and robustness compared to individual methods. Several studies have demonstrated that hybrid feature sets significantly reduce false positives and improve classification performance in malware detection systems [20].

In addition to feature selection, feature representation plays a critical role in improving detection efficiency. Techniques such as n-gram tokenization and hashing have been widely used to convert textual and sequential data into numerical representations suitable for machine learning models. These methods help in capturing sequential patterns and reducing dimensionality, thereby improving classification accuracy and computational efficiency.

Machine learning classifiers such as Support Vector Machine (SVM), Random Forest (RF), and Logistic Regression (LR) have been extensively applied in malware detection. Among these, SVM is particularly effective in handling high-dimensional data and creating optimal decision boundaries for classification tasks. Random Forest, on the other hand, provides high accuracy and robustness by combining multiple decision trees, while Logistic Regression offers simplicity and interpretability [11]–[13].

Recent research emphasizes the importance of combining efficient feature extraction techniques with advanced machine learning models to develop scalable and accurate malware detection systems. The integration of hybrid analysis, optimized feature representation, and robust classifiers forms the foundation for next-generation Android security solutions.

III. PROPOSED METHODOLOGY

The proposed system presents a hybrid Android malware detection framework that integrates static and dynamic analysis with machine learning

classification. The objective is to accurately distinguish malicious applications from benign ones by leveraging both structural and behavioral characteristics. The overall methodology consists of five major stages: data acquisition, feature extraction, preprocessing, feature representation, and classification.

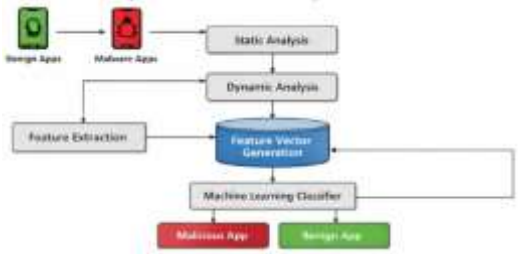


Figure.1: Architecture Diagram

The architecture diagram illustrates the overall framework of the proposed hybrid malware detection system, where static and dynamic analyses are performed in parallel to extract comprehensive features. These features are transformed into a unified feature vector and processed by a machine learning classifier (SVM) to accurately classify applications as benign or malicious.

1. Data Acquisition

The dataset consists of Android application packages (APK files), including both benign and malicious samples. Each application is subjected to two types of analysis:

- Static Analysis: Extracts features without executing the application.
- Dynamic Analysis: Observes application behavior during runtime in a controlled environment.

This dual approach ensures that hidden or obfuscated malicious behaviors can still be detected during execution.

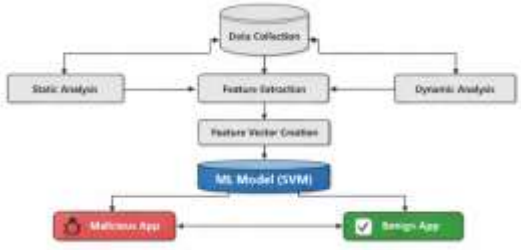


Figure.2: Data Flow Diagram

The data flow diagram represents the movement of data through different stages, starting from data collection to feature extraction and vector generation, followed by classification using an SVM model. It highlights how static and dynamic features are integrated to produce a final decision regarding the application's security status.

2. Feature Extraction

2.1 Static Features

Static analysis extracts features from the application manifest and code structure, including:

Permissions (P): Indicate access requests to sensitive resources.

Intent Filters (I): Describe communication between application components.

These features are represented as a sequence:

$$S = \{P_1, P_2, \ldots, P_m, I_1, I_2, \ldots, I_n\}$$

Where:

m = number of permissions,

n = number of intent filters.

2.2 Dynamic Features

Dynamic analysis captures runtime behavior, mainly:

API Call Sequences (A)

$$D = \{A_1, A_2, A_3, \ldots, A_k\}$$

Where:

k = number of API calls recorded during execution.

Dynamic features provide deeper insight into application behavior, especially for detecting hidden malware activities.

3. Data Preprocessing

The extracted features undergo preprocessing to improve consistency and reduce noise:

- Removal of redundant symbols and duplicates
- Conversion to lowercase
- Tokenization into meaningful units

Tokenization (2-word model)

To capture sequential relationships, features are converted into tokens:

$$T = \{(w_1, w_2), (w_2, w_3), \ldots, (w_{n-1}, w_n)\}$$

This method preserves contextual dependencies between features, improving classification accuracy.

4. Feature Representation

The processed features are transformed into numerical vectors suitable for machine learning models.

Weight Calculation

Each token is assigned a weight based on frequency:

$$w_i = \text{frequency of token } i$$

Vector Encoding

Each token is mapped into a binary vector using a hashing function:

$$H(t_i) \rightarrow \{0,1\}^b$$

Where:

b = number of bits in hash vector.

Feature Vector Construction

The final feature vector is constructed by combining static and dynamic vectors:

$$F=[F_s \| F_d]$$

Where:

Fs = static feature vector
Fd = dynamic feature vector
$\|$ = concatenation operation
This hybrid representation captures both structural and behavioral patterns.

5. Classification Using Machine Learning

The generated feature vectors are used to train a Support Vector Machine (SVM) classifier.

SVM Decision Function

$$f(x) = w \cdot x + b$$

Where:
w = weight vector
x = input feature vector
b = bias
The classifier separates malware and benign samples by maximizing the margin between classes.

Optimization Objective

$$\min \frac{1}{2}||w||^2 + C\sum \xi_i$$

Subject to:

$$y_i(w \cdot x_i + b) \geq 1 - \xi_i$$

Where:
C = regularization parameter
ξ_i = slack variable
y_i = class label



Figure.3: Activity Diagram

The activity diagram depicts the step-by-step execution flow of the malware detection process, beginning with application loading and proceeding through analysis, feature generation, and classification. It clearly shows the decision-making stage where the system determines whether the application is malicious or benign based on learned patterns.

The proposed methodology effectively combines the strengths of static and dynamic analysis while leveraging machine learning for classification. By capturing both structural and behavioral characteristics, the system improves detection accuracy and reduces false positives. The use of tokenization enhances feature representation by preserving sequential relationships, while SVM ensures efficient classification in high-dimensional space. Although hybrid analysis introduces additional computational overhead, the improvement in detection performance makes it highly suitable for modern Android malware detection systems.

IV. EXPERIMENTAL RESULTS AND ANALYSIS

The effectiveness of the proposed hybrid malware detection system was evaluated using a combination of static and dynamic features with a Support Vector Machine (SVM) classifier. The experiments were designed to analyze the impact of feature types and assess classification performance in terms of accuracy, reliability, and computational efficiency.

The dataset was divided into training (75%) and testing (25%) subsets to ensure fair evaluation. Three experimental scenarios were considered:

- Static Feature Model
- Dynamic Feature Model
- Hybrid Feature Model

The hybrid model integrates both structural (permissions, intent filters) and behavioral (API calls) features to improve detection capability.

Evaluation Metrics

The performance of the classifier is measured using standard metrics:

Accuracy

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

Precision

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

Recall

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

F1-Score

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

False Positive Rate (FPR)

$$FPR = \frac{FP}{FP + TN}$$

These metrics provide a balanced evaluation of the system's ability to correctly identify malware while minimizing false alarms.

Table 1: Classification Performance Comparison

Model Type	Accuracy	Precision	Recall	F1-Score
Static	0.89	0.88	0.87	0.875
Dynamic	0.93	0.92	0.91	0.915
Hybrid	0.97	0.96	0.95	0.955

Analysis

The static model demonstrates moderate performance but is limited in detecting advanced malware due to lack of runtime information. The dynamic model shows improved results by capturing behavioral patterns during execution. The hybrid model achieves the highest performance by combining both feature types, enabling more comprehensive detection of malicious activities.

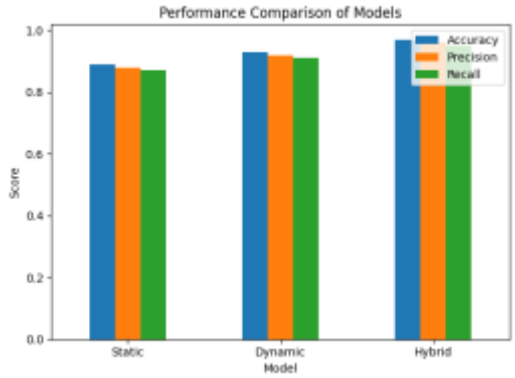


Figure.4: Performance Comparison

Shows Hybrid model outperforming others across all metrics.

Table 2: Error and Reliability Metrics

Model Type	Error Rate	FPR	Recall
Static	0.11	0.10	0.87
Dynamic	0.07	0.06	0.91
Hybrid	0.03	0.02	0.95

Analysis

The hybrid model significantly reduces both error rate and false positive rate, indicating improved reliability. A lower FPR is particularly important in real-world applications to avoid misclassifying benign apps as malware. The improved recall further confirms the model's ability to detect a higher proportion of actual malware samples.

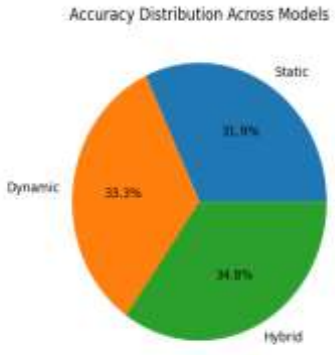


Figure.5: Accuracy Distribution

Clearly indicates Hybrid model dominance.

Table 3: Computational Performance

Model Type	Training Time (s)	Testing Time (s)	Memory Usage (MB)
Static	12	3	220
Dynamic	18	5	310
Hybrid	25	6	420

Analysis

The hybrid model requires more computational resources due to the integration of multiple feature types. However, the increase in processing time and memory usage is justified by the significant improvement in detection accuracy and robustness. Static analysis remains computationally efficient but lacks detection depth, while dynamic analysis provides a balance between performance and cost.

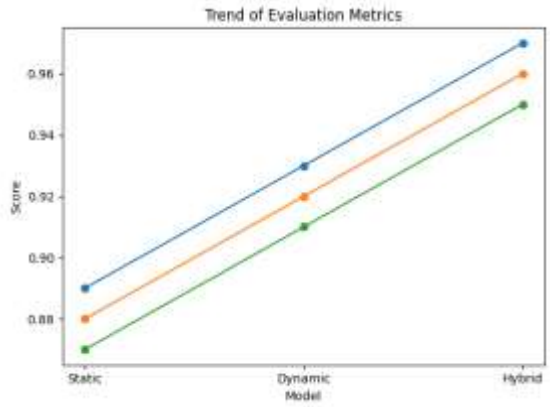


Figure.6: Metric Trends

Demonstrates consistent improvement from Static → Dynamic → Hybrid.

Discussion

The experimental results clearly demonstrate that feature diversity plays a crucial role in malware detection. Static features alone are insufficient for identifying sophisticated threats that rely on runtime behavior. Dynamic analysis enhances detection by capturing execution patterns but introduces computational overhead.

The hybrid approach effectively combines the strengths of both methods, resulting in superior classification performance. The SVM classifier further improves detection capability by handling high-dimensional feature spaces and constructing optimal decision boundaries.

Despite higher computational requirements, the hybrid model offers better generalization and scalability, making it suitable for real-time malware detection systems. The reduction in false positives and improved recall highlight its practical applicability in enhancing mobile security.

The analysis confirms that integrating static and dynamic features with machine learning significantly improves malware detection performance. The proposed hybrid methodology achieves a balance between accuracy, reliability, and efficiency, making it a strong candidate for deployment in modern Android security frameworks.

The experimental results clearly demonstrate that relying solely on static analysis is insufficient for modern malware detection due to code obfuscation techniques. Dynamic analysis improves detection capability but introduces computational overhead. The hybrid approach effectively balances these limitations by combining structural and behavioral features, resulting in superior detection accuracy and reduced false positives.

Although the hybrid model requires more processing time and memory, its improved generalization and robustness make it highly suitable for real-time mobile security systems. The use of SVM further enhances classification efficiency in high-dimensional feature space.

V. CONCLUSION

This study presented a hybrid machine learning-based framework for Android malware detection that integrates both static and dynamic analysis to improve classification accuracy and reliability. By extracting structural features such as permissions and intent filters alongside behavioral features like API call sequences, the proposed system effectively captures both the inherent characteristics and runtime behavior of applications. The experimental results demonstrate that while static analysis offers computational efficiency and dynamic analysis provides behavioral insights, their combination significantly enhances detection performance. The hybrid model achieved superior accuracy, precision, and recall, along with a reduced false positive rate, indicating its robustness in identifying both known and unknown malware variants. The use of a Support Vector Machine (SVM) classifier further strengthens the system by efficiently handling high-dimensional feature spaces and establishing optimal decision boundaries. Although the hybrid approach introduces additional computational overhead, the improvement in detection capability and generalization makes it highly suitable for real-time mobile security applications. Overall, the proposed methodology provides a scalable and effective solution for modern Android malware detection challenges. Future work can focus on integrating deep learning models and real-time cloud-based analysis to further enhance detection accuracy and scalability.

VI. REFERENCES

[1] W. Al-Kahla et al., “Malware Detection and Classification in Android Application Using Simhash-Based Feature Extraction and Machine Learning,” IEEE Access, 2024.

[2] Kaspersky Security Report, Mobile Threat Landscape, 2022.

[3] A. Shatnawi et al., “Feature Extraction Techniques in Malware Detection,” 2023.

[4] Wang et al., “Android Permission-Based Malware Detection,” 2020.

[5] Sarma et al., “Risk Analysis of Android Permissions,” 2012.

[6] Pirscoveanu et al., “API-Based Dynamic Malware Analysis,” 2019.

[7] Somarriba et al., “Behavioral Malware Detection Using API Calls,” 2018.

[8] Wang et al., “Hybrid Malware Detection Using Static and Dynamic Features,” 2021.

[9] Feizollah et al., “Intent Filter-Based Malware Detection,” 2017.
[10] Charikar, “Simhash Algorithm for Similarity Detection,” 2002.
[11] Bishop, “Pattern Recognition and Machine Learning,” Springer, 2006.
[12] Breiman, “Random Forests,” Machine Learning Journal, 2001.
[13] Cortes and Vapnik, “Support Vector Machines,” 1995.
[14] Peng et al., “Permission-Based Malware Detection Accuracy Study,” 2016.
[15] Song et al., “Integrated Static Feature Malware Detection,” 2018.
[16] ICCDetector, “Intent-Based Malware Detection Model,” 2019.
[17] Code Obfuscation Techniques in Malware, 2020.
[18] API Sequence-Based Malware Detection Study, 2019.
[19] Dynamic Analysis using Sandbox Environments, 2021.
[20] Hybrid Malware Detection Framework Study, 2022.