Sistemasi: Jurnal Sistem Informasi
Volume 14, Nomor 5, 2025: 2540-2551

ISSN:2302-8149
e-ISSN:2540-9719

Static Analysis-based Detection of Android Malware using Machine Learning Algorithms

[1]Omar Emad Saied*, [2]Karam H. Thanoon
[1]Department of Software, College of Computer Science and Mathematics, University of Mosul, Iraq
[2]Department of Cyber Security, College of Computer Science and Mathematics, University of Mosul, Iraq
*e-mail: [1]Omaremad_gold@uomosul.edu.iq, [2]karamhatim@uomosul.edu.iq

(***received***: 5 July 2025, ***revised***: 19 July 2025, ***accepted***: 20 July 2025)

Abstract

The rapid growth of Android applications has led to increased security threats, making malware detection a critical concern in cybersecurity. This research proposes a static analysis-based technique that employs machine learning for Android malware detection. The proposed method utilizes three classification algorithms: Support Vector Machine (SVM), Random Forest, and Decision Tree. The tool extracts static permission features from APK files to evaluate their effectiveness. The dataset consists of 400 Android applications (200 benign and 200 malicious), which were analyzed using the three machine learning models. Their performance was evaluated and compared using accuracy , precision, recall, and F1-score. The Random Forest model achieved the highest accuracy. The results demonstrate that static analysis combined with a robust classification model can effectively identify malicious applications with a high degree of accuracy. Although the tool is reliable in detecting Android malware, it has limitations in handling obfuscated and dynamic threats. Future research could focus on integrating dynamic analysis techniques to improve detection accuracy and enhance resistance to evasion techniques.

Keywords: *android malware, static analysis, machine learning, support vector machine, random forest, decision tree, cybersecurity*

1 Introduction

Software security involves protecting software from various types of malware and is a crucial aspect of cybersecurity. Initially, hacking was mainly considered a prank targeting victims' machines. However, with the continuous evolution of internet services the motivation behind cyberattacks shifted from mere amusement to financial gain and other valuable assets [1].

Mobile devices have become essential for various services such as communication, entertainment, financial transactions, and education. Software applications are now deeply integrated into daily life, influencing critical domains like traffic control, aviation, and self-driving cars [2]. For many people, life without these devices is unimaginable.

Android is one of the most popular and widely used mobile operating systems, consistently evolving and gaining popularity. Modern applications and devices such as smartphones, laptops, printers, and scanners have introduced several security challenges [3]. According to StatCounter, Android dominated the global smartphone operating system market in 2023, accounting for approximately 71.74% [4].

Because of its open-source nature and frequent updates by a vast developer community, Android has become a primary target for cyberattacks, mainly through malicious applications (malware). Android malware has evolved rapidly, adopting sophisticated techniques such as encryption and obfuscation to conceal its malicious intent.

Attackers can be individuals or organized groups, including former intelligence operatives or independent hackers. In general, hackers are categorized into two main groups: black-hat hackers, who exploit vulnerabilities for malicious purposes, and white-hat hackers, who work to improve system security [5].

Static analysis is one of the primary methods used to detect malware on Android devices. It involves extracting permission features from an APK application to identify malicious behavior, as shown in Figure 1. However, its major drawback is vulnerability to techniques such as code obfuscation and polymorphism, which complicate feature extraction [6].
Static and dynamic analysis operate differently but share the same goal of detecting malicious behavior. For instance, if an application uses location permissions to transmit a user's geolocation to a third-party server, both static and dynamic analysis would typically classify this as a privacy leakage case [7].

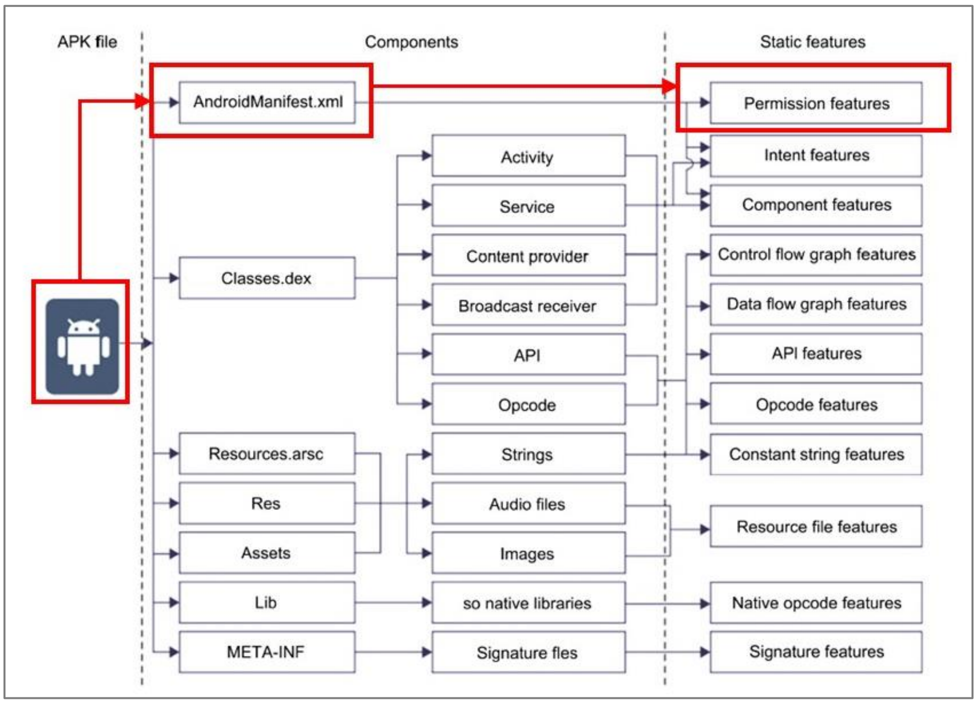


Figure 1 Taxonomy of Android malware features [8]

To address these challenges, various malware detection techniques have been developed, including static, dynamic, and hybrid approaches. Static analysis is considered efficient because it examines an application's code and structure without executing it [9]. By inspecting permissions, API calls, and control flow graphs, static analysis provides valuable information for identifying malicious applications. Its main advantage lies in detecting vulnerabilities before application installation and execution.
However, static analysis struggles with highly obfuscated or polymorphic malware, where attackers use techniques like variable renaming, junk code insertion, or code encryption to hide malicious intent [10].
One of the major problems related to detecting malware using just static analysis is their advanced evasion techniques, including obfuscation and polymorphism. Obfuscation makes the code of an application harder to understand. For example, it can be done through renaming a variable, inserting superfluous or misleading code, or encrypting parts of the application's code, which will then only be decrypted at runtime [8]. static feature extraction becomes ineffective and is often misclassified. Similarly, polymorphism also provides malware to modify its structure or signature on each infection while maintaining its malicious functionality, thereby drastically reducing the effectiveness of signature-based or static analysis methods [11].The above confer wishes the combination of static and dynamic techniques in the future to address the issue of such malware evasion and improve detection accuracy.
Recent research has focused on enhancing static analysis by integrating machine learning techniques, which can identify patterns and anomalies in application code more effectively.

This research aims to improve Android malware detection using static analysis combined with machine learning algorithms. Specifically, it investigates critical static features of permissions to develop a robust and scalable detection system capable of identifying both known and new malware variants.

2 RELATED WORK

Malware analysis has gained increasing attention due to the growing security threats targeting the Android system. Various methods have been proposed to detect malicious applications, including machine learning, static analysis, and dynamic analysis. This section discusses previous studies, highlighting their contributions, strengths, and weaknesses, in order to identify research gaps and improve Android security.

Malware analysis aims to provide sufficient information to assess the severity of malicious software. It involves examining malware to understand its behavior, evolution, targeted victims, and other relevant aspects [12]. The static analysis presented in [13] examines the structure of executable files without executing them, such as in the case of the Lime Worm Ransomware. This method involves extracting information such as hash codes, analyzing PE files using PeStudio, and identifying suspicious string patterns. Signature verification is performed using VirusTotal to determine whether a file is malicious.

There are many studies that employ static analysis to develop machine learning models capable of differentiating between benign and malicious applications. A number of these studies focused on permission-based classification, opcode sequence detection, and API usage analysis. Reverse engineering was also done to access internal components of APK files (like .dex, resources, and manifest).

According to the paper in [14], a new mechanism for feature selection is suggested independent of dynamic analysis, various different features such as API calls and permissions are extracted and classified using machine learning techniques. The primary aim of this method is to enhancement detection accuracy by lowering number of features. In a similar context, the paper in [15] applies static analysis to extract and analyze the features of the application before execution by examining APK files. Dimensionality reduction is achieved through factor Analysis of permissions, API calls, opcodes, and textual data from the manifest file followed by Broad Learning to improve prediction accuracy. The The research based in [16] uses static code from an application in order to extract function call graphs. Merging this with dynamic features, this approach significantly improves accuracy detection, especially against dynamically loaded code and obfuscation, which pose major challenges in detecting malware.

In addition, in research [17] collects features such as permissions, API calls, and URLs, then leverages large language models (LLMs) to analyze relationships between features, accuracy detection is enhanced, and detailed interpretability reports are provided. In [18], detecting riskware by analyzing APK files without execution. Extracting features includes permissions, API calls, manifest entries, and coding structures. These features help to identify high-risk permissions, sensitive API access, dynamic code loading, and obfuscation patterns, which are then fed into machine learning models for riskware classification.

3 Dataset Creation

A custom-built dataset of Android applications was specifically constructed for this study for enable the analysis of malicious behavior and detection. This dataset combines critical features such as application permissions, which play an important role in supporting the process of classification. It was organized to arranged to make it compatible with machine learning techniques.

The Kronodroid dataset from [14] is used for static analysis, where features such as API calls, permissions, and opcode instructions are extracted from the applications without executing. Feature selection techniques were used to find the most influential factors that cause harmful activity to improve classification accuracy in static analysis. Moreover, this method decreases the number of features for classification which enhances processing efficiency.

ISSN:2302-8149
e-ISSN:2540-9719

Android applications, as shown in Table 1, were downloaded from trusted sources such as the Google Play Store [19], Uptodown, and APKPure for benign samples. Malicious applications were obtained from reputable global platforms that legally provide malware samples for research purposes, supporting the training and development of Android malware detection tools. Key sources included AndroZoo [15] and VirusTotal [10], which supplied malicious APK samples.
The resulting dataset was balanced, consisting of an equal number of benign and malicious applications. This balanced composition was specifically designed to facilitate the extraction of permission-related features, thereby enabling accurate classification between benign and malicious applications.

Table 1 Source of Benign and Malicious APK

Source Name	Website URL	Label	Number of Samples
Google play	https://play.google.com/store	Benign	55
APKpure	https://apkpure.com/en/	Benign	75
Uptodown	https://en.uptodown.com/	Benign	70
Vrusetotal	https://www.virustotal.com/gui/home/upload	Malicious	88
MalwareBazaar	https://bazaar.abuse.ch/	Malicious	68
AndroZoo	https://androzoo.uni.lu/	Malicious	44
	Total		400

4 Methodology

The primary objective of this research is to analyze and detect Android malware applications based on the permissions requested by an application without executing it, as illustrated in Figure 2. To achieve this goal, malware was classified according to its infection mechanisms, enabling the identification of the types of malware that the proposed methodology can detect at the permission level.

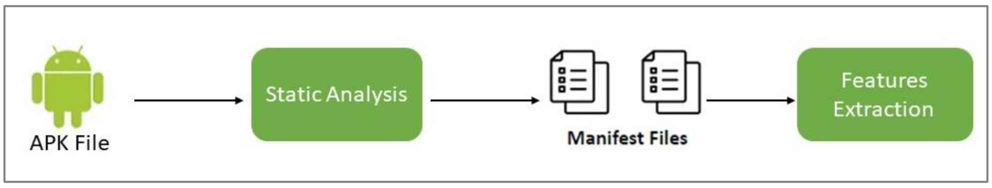


Figure 2 Static analysis design

Initially, the Android application file is processed through the proposed tool, which performs static analysis by examining the application’s code and manifest files without executing it, as illustrated in Figure 3. The static analysis focuses on extracting requested permissions features, which are essential indicators for identifying malicious behaviors. To enhance detection accuracy , these extracted features undergo feature selection techniques to identify the most discriminative attributes for malware classification. The study in [20] supports this approach by analyzing malicious files to determine the most significant behavioral patterns and infection strategies, which aids in improving feature-based classification. All selected features are systematically stored in an Excel sheet for further processing and model training.
In the subsequent step, the extracted features are fed into a machine learning (ML) algorithm to classify the application as either benign or malicious. Essential machine learning techniques, including classification, regression analysis, clustering, feature engineering, and deep learning, are employed to enhance prediction accuracy and improve classification performance [21]. To further optimize classification results, a feature selection technique is applied to identify the most relevant features. For instance, as demonstrated in [22], an SVM-based model achieved highly accurate

ransomware detection, where static features yielded an accuracy of 81%, while dynamic features achieved 100% accuracy.

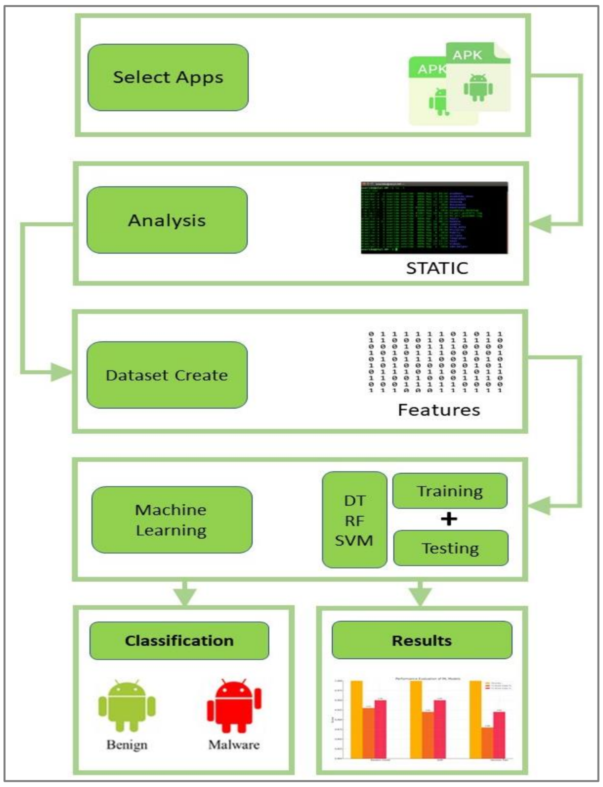


Figure 3 The proposed tool works

5 Experimental Setup and Static Feature Extraction Process

The experiments were conducted using Android Studio Ladybug 2024.2.1 Patch 3 on a Windows 11 Home 64-bit system. Static analysis was employed for malware detection, and the implementation was performed in Python using the following libraries: Androguard for APK file analysis, ADB (Android Debug Bridge) for accessing application files, and Scikit-learn for implementing machine learning algorithms. The custom dataset used in this study consisted of 400 balanced applications.
The proposed tool analyzes Android applications through static analysis, meaning that the app file is examined without executing it on a physical device or emulator. The main objective is to identify the requested permissions, which are key indicators of an application's intended behavior and potential impact on the system.
The static analysis process consists of the following steps:

1. Extracting the AndroidManifest.xml by Unpacking APK or XAPK files, and all application declarations requested permissions are registered.

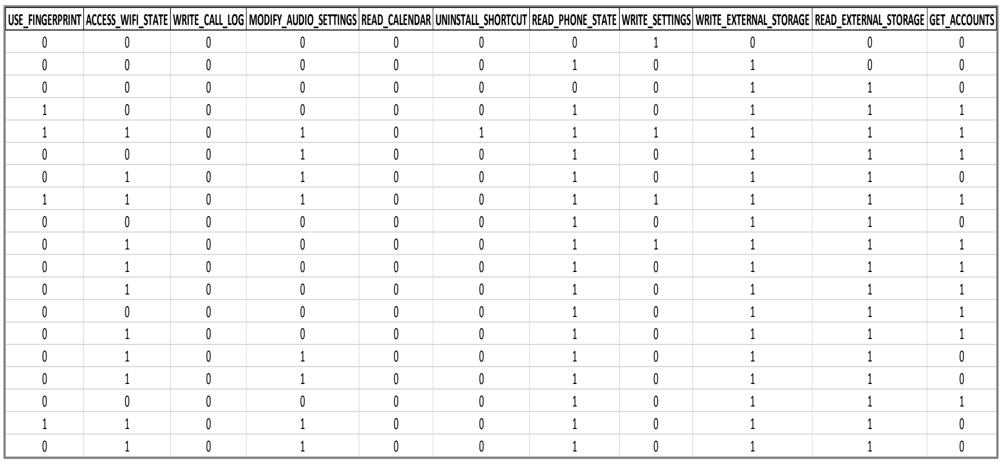
2. Analyzing the application structure and retrieving the complete list of requested permissions.

3. Extracted permissions are transforming them into a machine-readable format by converting them into a binary representation (1 = permission requested, 0 = permission not requested).

4. Constructing a feature table, where each application is represented as a row and each column represents a specific permission or feature.

5. Labeling the applications for classification, where malicious applications are assigned the value 1, while benign applications are assigned the value 0.

After these operations are completed, extracted features are stored in the database, as shown in Figure 4, and are prepared for machine learning classification to distinguish between benign and malicious applications. then, applied feature selection techniques to identify the most relevant permissions, and improving classification performance while reducing computational complexity.



USE_FINGERPRINT	ACCESS_WIFI_STATE	WRITE_CALL_LOG	MODIFY_AUDIO_SETTINGS	READ_CALENDAR	UNINSTALL_SHORTCUT	READ_PHONE_STATE	WRITE_SETTINGS	WRITE_EXTERNAL_STORAGE	READ_EXTERNAL_STORAGE	GET_ACCOUNTS
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0	1	0	0
0	0	0	0	0	0	0	0	1	1	0
1	0	0	0	0	0	1	0	1	1	1
1	1	0	1	0	1	1	1	1	1	1
0	0	0	1	0	0	1	0	1	1	1
0	1	0	1	0	0	1	0	1	1	0
1	1	0	1	0	0	1	1	1	1	1
0	0	0	0	0	0	1	0	1	1	0
0	1	0	0	0	0	1	1	1	1	1
0	1	0	0	0	0	1	0	1	1	1
0	1	0	0	0	0	1	0	1	1	1
0	0	0	0	0	0	1	0	1	1	1
0	1	0	0	0	0	1	0	1	1	1
0	1	0	1	0	0	1	0	1	1	0
0	1	0	1	0	0	1	0	1	1	0
0	0	0	0	0	0	1	0	1	1	1
1	1	0	1	0	0	1	0	1	1	0
0	1	0	1	0	0	1	0	1	1	0

Figure 4 Dataset after feature extraction

6 Results & Performance Analysis

Three widely used machine learning classification algorithms applied in this research: Random Forest (RF), Support Vector Machine (SVM), and Decision Tree (DT) to distinguish benign from malicious Android applications. Evaluating the performance of these models by using Accuracy, Precision, Recall, and F1-Score for both Class 0 (benign) and Class 1 (malicious), as shown in Table 2. The aim of this section is to assess the effectiveness of the developed tool in detecting malware applications through comparing its performance against three other approaches to determine whether it performs better [23].
The highest accuracy was achieved by Random Forest (93.75%), showing balanced recall and precision scores. SVM also performed well, achieving high results, although slightly lower than Random Forest in malware detection. Decision Tree produced the lowest performance. Based on these findings, we recommended Random Forest as the best model for Android malware detection because its reduces overfitting, has high accuracy, and robust performance.

6-1 Random Forest (RF)

It is an ensemble learning method for building multiple decision trees and combining their predictions by voting mechanism, which reduces overfitting and enhances accuracy , It is, in particular effective for high-dimensional datasets. RF achieved an overall accuracy of 93.75%. For Class 0 (benign), the model acquired Precision of 0.91, Recall of 0.94, and F1-Score of 0.93, strong ability to classify benign applications correctly. For Class 1 (malicious), the model performed even better, achieving a Precision of 0.96, Recall of 0.94, and F1-Score of 0.95.

6-2 Support Vector Machine (SVM)

It is a popular classification algorithm which finds the optimal hyperplane to separate classes in a multi-dimensional space. performing effectively in high-dimensional and non-linear datasets when using appropriate kernels. In this research, the analysis of SVM achieved an overall accuracy like to RF (93.75%). For Class (0) it achieved a Precision of 0.94, Recall of 0.91 , and F1-Score of 0.92. For Class (1) the scores were Precision: 0.94 , Recall: 0.96 , and F1-Score: 0.95. indicating these results, while SVM is high effective in malicious applications detection, it performs lower effectively in identifying benign applications compared to RF.

6-3 Decision Tree (DT)

It is an essential classification algorithm that division datasets into branches (nodes) based on feature thresholds. It is simple to implement and interpret, and prone to overfitting, exceptionally when pruning or parameter tuning is not applied. In this work, DT achieved an overall accuracy of 90.00% . For Class (0) the model scored a Precision of 0.90, Recall of 0.85 , and F1-Score of 0.88 , indicating a notable decline in correctly identifying benign applications. For Class (1) it achieved a Precision of 0.94, Recall of 0.90, and F1-Score of 0.92, which is slightly lower compared to RF and SVM.

Table 2 Performance evaluation

Model	**Accuracy**	Class 0			Class 1		
		Precision	**Recall**	**F1-Score**	**Precision**	**Recall**	**F1-Score**
RF	93.75	0.91	0.94	0.93	0.96	0.94	0.95
SVM	93.75	0.94	0.91	0.92	0.94	0.96	0.95
DT	90.00	0.90	0.85	0.88	0.90	0.94	0.92

7 Evaluation Metrics

The evaluation metrics presented in Table 3 are essential for assessing the performance of a malware classification model based on machine learning. Ensuring that an efficient malware detection model achieves high precision to lower false positives (FP) and high recall to minimize false negatives (FN) while keeping an optimal balance between these two measures through the F1-score.

Precision is defined as the ratio of correctly identified positive instances (true positives) to all instances predicted as positive (true positives plus false positives). Recall (or sensitivity) measures the proportion of actual positive instances that are correctly identified by the model . The F1-score combines precision and recall into a single value providing a balanced evaluation of the system’s overall performance. A maximum value of 1.000 indicates optimal detection performance [24].

The performance of the proposed tool was evaluated using the following metrics :

- Accuracy: The percentage of applications correctly classified as either benign or malicious.
- Precision: The ability of the model to correctly identify malware while minimizing false alarms.
- Recall: The capability of the model to correctly identify the majority of malicious applications.
- F1-score: A harmonic mean of precision and recall, providing a single measure for balanced evaluation.

Confusion Matrix : A detailed breakdown of correct and incorrect classifications, used to analyze the distribution of true positives , false positives , true negatives , and false negatives.

Table 3 Description of evaluation metrics.

Metrics	**Descriptions**
$T\ P$	The number of correctly identified malicious Apps.
TN	The number of correctly identified benign Apps.
FP	The number of misidentified benign Apps.
TN	The number of misidentified malicious Apps.
ACC	$(T\ P + TN)/(T\ P + TN + FP + FN)$
$Precision$	$T\ P/(T\ P + FP\ )$
$Recall$	$T\ P/(T\ P + FN)$
$F1$	$(2 \times Precision \times Recall)/(Precision + Recall)$

ISSN:2302-8149
e-ISSN:2540-9719

8 Result Discussion

The results of the classification comparison indicate that the Random Forest (RF) model achieved the highest accuracy (93.75%) equal to that of the Support Vector Machine (SVM) classifier, as illustrated in Figure 5. However, RF outperformed the other classifiers in terms of the F1-score for both Class 0 (benign) and Class 1 (malicious) , establishing it as the most effective model overall.

Contrary to some expectations, RF demonstrated strong capabilities in detecting both benign and malicious applications, with no significant weaknesses in identifying malicious samples. Although the SVM achieved the same accuracy (93.75%) as RF, it showed slightly better precision for Class 0 (RF: 0.91 vs. SVM: 0.94). However, its F1-score for Class 0 was slightly lower (RF: 0.93 vs. SVM: 0.92). Notably , SVM achieved the highest recall for Class 1, reaching 0.96, indicating superior performance in detecting malicious applications.

The Decision Tree (DT) recorded the lowest accuracy (90.00%) among all classifiers. Its weakest performance was observed in the recall for Class 0 (0.85) suggesting difficulties in correctly identifying benign applications. DT showed competitive performance in detecting Class 1, with precision (0.94) and recall (0.90) values comparable to those of RF and SVM.

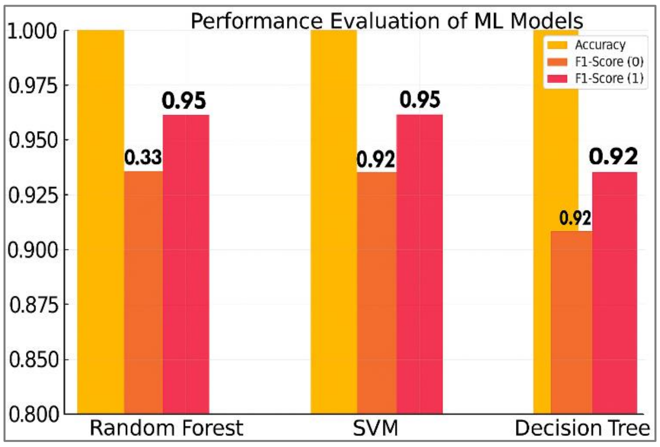


Figure 5 Comparative performance of classification algorithms

the confusion matrix as shown in Figure 6 a value of 1 indicates a positive classification (malicious application), while a value of 0 indicates a negative classification (benign application). higher concentration of values along the diagonal of the confusion matrix demonstrates the model’s effectiveness , as it reflects a high number of correct classifications for both benign and malicious applications.

ISSN:2302-8149
e-ISSN:2540-9719

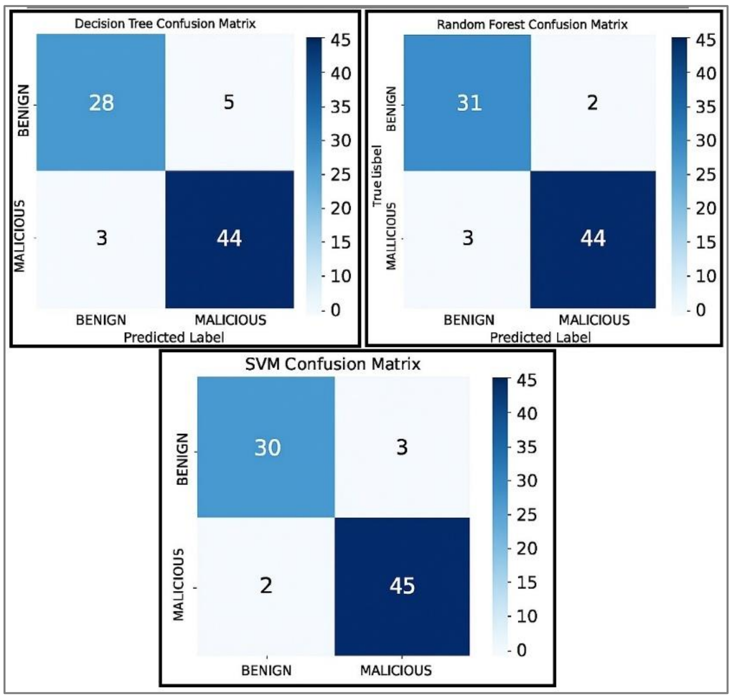


Figure 6 Confusion matrix of all 3 ML malware detection models.

9 Performance Comparison

A custom dataset was specifically developed for this research, which presents certain challenges in establishing direct comparisons with other studies. While most related works utilize publicly available benchmark datasets such as AndroZoo, VirusTotal, or Kaggle, which are widely used and easily accessible to the research community, the dataset in this study was constructed using live APK applications.
Permissions were extracted in a real-world context, providing more realistic and practically relevant data for malware detection. However, this approach introduces limitations in terms of direct comparability with the results reported in previous literature.
Table 4 presents a comparative analysis between the findings of this research and the most recent, closely related studies that share similar objectives, providing context for evaluating the performance of the proposed methodology.

Table 4 Comparative study with related work.

Ref.	Feature used	Classification algorithm	Classification performance	date
[25]	Permissions, commands, function calls	Random Forest	90.47% (using permissions only)	2024
[26]	Permission-based static features	Random Forest and other classifiers	Random Forest TPR: 91.6%	2021
This Research	Permission-based static features	RF	93.75%	2025
		SVM	92.63%	
		DT	90.00%	

10 Challenges & Limitations

Despite the promising results, several limitations may influence the performance of the proposed tool. First, the analysis was conducted entirely using static techniques, which limits the tool’s ability to detect malware that relies on dynamic behavior. Additionally , the tool demonstrates limited capability in identifying benign applications that can rapidly transform into malware through obfuscation techniques [27].

The tool may also fail to function effectively with applications that deliberately obscure or encrypt their code, a common strategy used by advanced malware to evade detection. Furthermore, the evaluation was conducted on a relatively small dataset of 400 applications , which may not fully reflect real-world performance.

In real-time applications, however, the Random Forest algorithm is expected to perform more efficiently due to its higher accuracy and robustness, making it a strong candidate for deployment in practical malware detection systems.

11 Conclusion and future work

Static analysis plays a crucial role in detecting and classifying Android applications by examining their code structure, permissions, API calls, and other essential features without requiring execution. This research investigated the effectiveness of static analysis techniques in detecting malicious applications. By analyzing manifest files, API usage, and other key static features, the study demonstrated that static analysis provides a proactive approach to malware detection.

The findings confirm that static features can be effectively used for early threat detection. However, they also highlight certain limitations, particularly against sophisticated malware that employs code obfuscation or dynamic behavior, where dynamic analysis becomes essential. The processed features in this static investigation, especially API calls and manifest files, continue to be preferred by security experts, underlining their significance in Android malware detection [27].

As part of future work, efforts are directed toward enhancing the static analysis component to extract a broader range of features beyond permissions, thereby improving classification accuracy. In parallel, the development of a dynamic analysis framework is underway, where applications will be executed in a controlled environment to monitor requested permissions, intents, and behavioral patterns during runtime, providing a more comprehensive detection mechanism against advanced and evasive malware.

ACKNOWLEDGMENT

We would like to thank the University of Mosul/Iraq and the College of Computer Science and Mathematics as well as the Department of Software Science for providing support and computer labs.

REFERENCE

[1] K. H. Thanoon, B. Mahmood, and M. M. A. Dabdawb, “The Effect of Malware’s Apis Relations on Software Security Design,” MINAR International Journal of Applied Sciences and Technology, Vol. 4, No. 1, pp. 1–157, Mar. 2022, doi: 10.47832/2717-8234.10.14.
[2] S. Hasoon, T. Najim AL-Hadidi, and S. Omar Hasoon, “Software Defect Prediction using Extreme Gradient Boosting (XGBoost) with Optimization Hyperparameter,” Journal of Computer Sciences and Mathematics (RJCM), Vol. 18, No. 1, pp. 22–29, 2024, doi: 10.33899/CSMJ.2023.142739.108.
[3] M. A. A. Al-Ameri, B. Mahmood, B. Ciylan, and A. Amged, “Unsupervised Forgery Detection of Documents: A Network-Inspired Approach,” Electronics (Switzerland), Vol. 12, No. 7, Apr. 2023, doi: 10.3390/electronics12071682.
[4] A. Muzaffar, H. R. Hassen, H. Zantout, and M. A. Lones, “DroidDissector: A Static and Dynamic Analysis Tool for Android Malware Detection,” Aug. 2023, doi: 10.1007/978-3-031-40598-3_1.
[5] C. S. Yadav et al., “Malware Analysis in IoT & Android Systems with Defensive Mechanism,” Electronics (Switzerland), Vol. 11, No. 15, Aug. 2022, doi: 10.3390/electronics11152354.
[6] I. Almomani, M. Ahmed, and W. El-Shafai, “Android Malware Analysis in a Nutshell,” PLoS One, Vol. 17, No. 7 July, Jul. 2022, doi: 10.1371/journal.pone.0270647.
[7] T. Tu, H. Zhang, B. Gong, D. Du, and Q. Wen, “Intelligent Analysis of Android Application

Privacy Policy and Permission Consistency," Artif Intell Rev, Vol. 57, No. 7, Jul. 2024, doi: 10.1007/s10462-024-10798-z.

[8] Q. Wu, X. Zhu, and B. Liu, "A Survey of Android Malware Static Detection Technology based on Machine Learning," 2021, *Hindawi Limited*. doi: 10.1155/2021/8896013.

[9] Z. Muhammad, Z. Anwar, A. R. Javed, B. Saleem, S. Abbas, and T. R. Gadekallu, "Smartphone Security and Privacy: A Survey on APTs, Sensor-based Attacks, Side-Channel Attacks, Google Play Attacks, and Defenses," Jun. 01, 2023, MDPI. doi: 10.3390/technologies11030076.

[10] S. Arshad, M. Ali, A. Khan, and M. Ahmed, "Android Malware Detection & Protection: A Survey," International Journal of Advanced Computer Science and Applications, Vol. 7, No. 2, 2016, doi: 10.14569/ijacsa.2016.070262.

[11] P. Faruki, R. Bhan, V. Jain, S. Bhatia, N. El Madhoun, and R. Pamula, "A Survey and Evaluation of Android-based Malware Evasion Techniques and Detection Frameworks," Jul. 01, 2023, Multidisciplinary Digital Publishing Institute (MDPI). doi: 10.3390/info14070374.

[12] N. faith M Jameel and M. M. T. Jawhar, "A Survey on Malware Attacks Analysis and Detected," International Research Journal of Innovations in Engineering and Technology, Vol. 07, No. 05, pp. 32–40, 2023, doi: 10.47001/irjiet/2023.705005.

[13] N. A. Sultan, K. H. Thanoon, and O. A. Ibrahim, "Ethical Hacking Implementation for Lime Worm Ransomware Detection," in Journal of Physics: Conference Series, Institute of Physics Publishing, May 2020. doi: 10.1088/1742-6596/1530/1/012078.

[14] S. Sharma, Prachi, R. Chhikara, and K. Khanna, "A Novel Feature Selection Technique: Detection and Classification of Android Malware," Egyptian Informatics Journal, Vol. 29, Mar. 2025, doi: 10.1016/j.eij.2025.100618.

[15] K. Kılıç, İ. Atacak, and İ. A. Doğru, "FABLDroid: Malware Detection based on Hybrid Analysis with Factor Analysis and Broad Learning Methods for Android Applications," Engineering Science and Technology, an International Journal, Vol. 62, Feb. 2025, doi: 10.1016/j.jestch.2024.101945.

[16] J. Feng, L. Shen, Z. Chen, Y. Lei, and H. Li, "HGDetector: A Hybrid Android Malware Detection Method using Network Traffic and Function Call Graph," Alexandria Engineering Journal, Vol. 114, pp. 30–45, Feb. 2025, doi: 10.1016/j.aej.2024.11.068.

[17] W. Zhao, J. Wu, and Z. Meng, "AppPoet: Large Language Model based Android Malware Detection via Multi-View Prompt Engineering," Apr. 2024, [Online]. Available: http://arxiv.org/abs/2404.18816

[18] M. M. Alani and M. Alawida, "Behavioral Analysis of Android Riskware Families using Clustering and Explainable Machine Learning," Big Data and Cognitive Computing, Vol. 8, No. 12, Dec. 2024, doi: 10.3390/bdcc8120171.

[19] H. INAYOSHI Supervisor and S. Saito, "A Study on Taint Analysis with Runtime Data for Tracking Information Flows in Android Apps," 2024.

[20] M. F. Ismael and K. H. Thanoon, "Investigation Malware Analysis Depend on Reverse Engineering using IDAPro," in 2022 8th International Conference on Contemporary Information Technology and Mathematics, ICCITM 2022, Institute of Electrical and Electronics Engineers Inc., 2022, pp. 227–231. doi: 10.1109/ICCITM56309.2022.10031698.

[21] A. Ali and N. N. Saleem, "Classification of Software Systems Attributes based on Quality Factors using Linguistic Knowledge and Machine Learning: A review.," Journal of Education and Science, Vol. 31, No. 3, pp. 66–90, Sep. 2022, doi: 10.33899/edusj.2022.134024.1245.

[22] H. Ngirande, M. Muduva, R. Chiwariro, and A. Makate, "Detection and Analysis of Android Ransomware using the Support Vector Machines," Int J Res Appl Sci Eng Technol, Vol. 12, No. 1, pp. 241–252, Jan. 2024, doi: 10.22214/ijraset.2024.57885.

[23] H. Babbar, S. Rani, D. K. Sah, S. A. AlQahtani, and A. Kashif Bashir, "Detection of Android Malware in the Internet of Things Through the K-Nearest Neighbor Algorithm," Sensors, Vol. 23, No. 16, Aug. 2023, doi: 10.3390/s23167256.

[24] D. Aboshady, N. Ghannam, E. Elsayed, and L. Diab, "The Malware Detection Approach in the Design of Mobile Applications," Symmetry (Basel), Vol. 14, No. 5, May 2022, doi: 10.3390/sym14050839.

[25] P. Sivaprakash, M. Sankar, J. Vimala Ithayan, and C. Ramalingam, "Autonomous Android Malware Detection System based on Static Analysis," in Proc. 2024 Int. Conf. Integration

Emerging Technol. Digital World (ICIETDW), Sep. 2024, pp. 1–6, doi: 10.1109/ICIETDW61607.2024.10939283.

[26] J. M. Arif, M. F. A. Razak, S. Awang, S. R. T. Mat, N. S. N. Ismail, and A. Firdaus, "A Static Analysis Approach for Android Permission-based Malware Detection Systems," PLoS One, Vol. 16, No. 9 September, Sep. 2021, doi: 10.1371/journal.pone.0257968.

[27] R. Jusoh, A. Firdaus, S. Anwar, M. Z. Osman, M. F. Darmawan, and M. F. A. Razak, "Malware Detection using Static Analysis in Android: a review of FeCO (Features, Classification, and Obfuscation)," PeerJ Comput Sci, Vol. 7, pp. 1–54, 2021, doi: 10.7717/peerj-cs.522.