SAJSET
ISSN: 2992-3581

Savannah Journal of Science and Engineering Technology
Volume: 03; Issue: 06; Pages: 615-620; 2025.
www.sajsetjournal.com.ng

DOI: xx.xxxx.x/sajset-03-2025-0086

SAJSET
ISSN: 2992-3581

Android malware detection using machine learning based on integrated static and dynamic features

[1]Liman, A. I. and [1]Ahmad, A. A.
[1]Department of Computer Science, Faculty of Science, Gombe State University, Gombe State, Nigeria
aaminu3@gsu.edu.ng

Paper History
Received: 01st Jan, 2023
Accepted: 14th Nov., 2025
Published: November, 2025

Corresponding author
Liman, I. A.
ibrahimahmadliman@gmail.com

Abstract:
The motivation behind this study arises from increasing rate of Android malware attacks, which threaten user privacy, data security, and financial safety. Existing detection systems often fail to identify newly developed or obfuscated malware, creating a need for a more intelligent and adaptive solution. This study addresses the growing threat of Android malware caused by the rapid rise of mobile applications and the limitations of existing detection methods. Traditional approaches such as signature-based and heuristic-based detection often fail to recognize new or obfuscated malware, leading to high false positives. To address these challenges, this research proposes a machine learning–based framework that integrates both static and dynamic features for malware detection. The system was trained and tested on a balanced dataset of 500 Android applications, consisting of 250 benign apps and 250 malware samples. The system extracts key attributes such as permissions, API calls, system calls, and network behaviors, and applies classifiers including Support Vector Machine (SVM), K-Nearest Neighbor (KNN), Naïve Bayes, Random Forest, and Multi-Layer Perceptron (MLP). Experimental results show that dynamic features provide higher accuracy than static features alone, while combining both improves robustness. Among the models, Random Forest achieved the best performance with an F1-score of 84.08% in the dynamic phase and 68.53% in the combined setting. The findings confirm that integrating static and dynamic features significantly enhances the accuracy and reliability of Android malware detection. This approach can assist mobile device users and cyber security firms in preventing malware attacks.

Keywords: *Android malware, Machine learning, Static analysis, Dynamic analysis.*

1. Introduction

The rapid expansion of mobile applications has positioned Android as the most widely used operating system worldwide. However, this popularity has also made Android devices a major target for cybercriminals, as malware can steal sensitive information, disrupt services, and compromise user privacy as reported by Suarez-Tangil [1]. Traditional detection methods, such as signature-based and heuristic approaches, have well-documented limitations: signature-based detection is effective only against known threats, while heuristic techniques often generate high false positives as reported by Gandotra [2].

To address these shortcomings, researchers have investigated two primary analysis techniques: static and dynamic analysis. Static analysis inspects application code without execution, enabling faster results but struggling with obfuscation and hidden logic. Barrera [3]. In contrast, dynamic analysis monitors runtime behavior, such as system calls and network activity, providing richer insights but requiring high computational resources and facing evasion attempts through anti-analysis techniques, as reported by Arora and Peddoji [4].

Recent literatures suggest that combined approaches—integrating static and dynamic features with machine learning—offer greater robustness, as they combine the strengths of both methods while mitigating their individual weaknesses. For example, Arora and Peddoju [4] proposed NTPDroid, which integrated permissions and network traffic. Kabakus and Drgru [5,] applied hybrid techniques for in-depth malware analysis, while Liu [6] demonstrated improved efficiency through static-dynamic integration with PCA. Also, some of the recent studies as reported by Haq and Khuthaylah [7] and Alhussen [8] have explored hybrid and deep-learning methods for Android malware detection. For example, "Hybrid and ensemble techniques using both static and dynamic features have shown very high accuracy Suarez-Tangil [1].Despite these contributions, many studies rely on limited feature sets or a narrow range of classifiers.

This study extends prior work by integrating both code-level attributes (permissions, API calls) and

behavioral data (system calls, network traffic, and resource usage). Multiple machine learning classifiers- SVM, KNN, Naïve Bayes, Random Forest, and MLP- were evaluated, and results show that dynamic features outperform static ones, and that combined features further enhances robustness. Among the models, Random Forest consistently delivered the best performance, confirming its suitability for Android malware detection.

Table1: Comparison of selected related studies on android malware detection

Study	Approach	Feature Used	Technique/Model	Key Findings/ Limitations
Barrera, *et al.* [3]	Static	Permissions	Empirical Analysis	Showed limitations of permission-based models; vulnerable to obfuscation
Arp, *et al.* [11]	Static	Permissions, API calls	Drebin (linear SVM)	High accuracy, but weak against obfuscation and dynamic behavior
Yerima, *et al.* [13]	Dynamic	System calls + Permissions	Parallel ML classifiers	Improved accuracy, but required heavy resources.
Arora and Peddoju [4]	Hybrid	Permission + Network traffic	NTPDroid	Robust hybrid detection, but limited dataset size
Liu, *et al.* [6]	Hybrid	Static + Dynamic	PCA + ML classifiers	Efficient and accurate, but evaluation lacked large-scale apps
Karbab, *et al.* [14]	Dynamic	Network traffic patterns	Deep learning (DL-Droid)	Strong runtime detection, but resource-intensive
Kabakus and Dogru [5]	Hybrid	Static + Dynamic	Hybrid machine learning	Achieved in-depth detection, but no real time testing
This study	Static + Dynamic	Permissions, API calls, system calls, network traffic	SVM, KNN, NB, RF, MLP	Balanced static-dynamic integration; Random Forest achieved best F1-score

2. Methodology

This study employed an experimental research designed to develop and evaluate an Android malware detection framework using static and dynamic features. The methodology consisted of dataset preparation, feature extraction, model development, and performance evaluation. The current study feature-selection approach is inspired by recent work that combined permission, API, and contextual features reported by Reddy [9]. For example, we extend the feature-vector strategies used by Reddy [9] to also include dynamic behavioral features. A systematic review identified over 100 ML-based Android malware detection studies, highlighting the most common features and challenges Senanayake [10]. For example, Senanayake [10] provided a broad overview of ML approaches, feature extraction methods, and open datasets

2.1. Dataset collection

A balanced dataset of 500 Android applications was compiled, consisting of 250 benign apps collected from the Google Play Store and open-source repositories such as APKMirror, and F-Droid while 250 malware samples were sourced from the Drebin dataset according to Arp [11] and other public malware repositories like AndroZoo. This ensured a representative set of applications for effective training and evaluation. The malware samples represented diverse families, including Trojans, spyware, and ransomware.

2.2. Feature extraction and preprocessing

Static features—including permissions, API calls, and intent filters—were extracted from APK files without execution. Dynamic features—including system calls, network traffic, and resource utilization—were collected by executing apps in a controlled sandbox environment Arora and Peddoji [4].

2.3. Data preprocessing

To enhance the performance of the classifiers, several preprocessing techniques were applied. First, the extracted features were encoded and normalized to uniform value scales across all attributes. Subsequently, the Synthetic Minority Over-sampling Technique (SMOTE) was employed to address class imbalance by generating synthetic samples for the minority class, thereby minimizing bias during model training. Finally, feature selection was performed using Information Gain (IG) for static features and Principal Component Analysis (PCA) for dynamic features to reduce dimensionality while preserving the most informative attributes. The workflow of the proposed framework is illustrated in Figure 1.

2.4. Machine learning models

Five machine learning classifiers – Support Vector Machine (SVM), K-Nearest Neighbor (KNN), Naïve Bayes (NB), Random Forest (RF), and Multilayer Perceptron (MLP) - were implemented to evaluate the performance of the proposed framework as reported by Alzaelaee [12].

a. Support Vector Machine (SVM) was used for its capability to construct optimal hyperplanes that separate classes effectively in high-dimensional spaces.
b. K-Nearest Neighbor (KNN) algorithm was included as a simple yet effective instance-based learner that classifies samples based on proximity to their nearest neighbors.
c. Naïve Bayes (NB) classifier, known for its probabilistic approach and computational efficiency, was employed to evaluate performance on feature sets with independence assumptions.
d. Random Forest (RF), an ensemble of decision trees, was selected for its robustness, ability to handle large feature spaces, and resistance to overfitting.

e. Finally, Multi-Layer Perceptron (MLP), a feed-forward artificial neural network, was used to capture complex nonlinear relationships between features.

These classifiers were chosen to provide a diverse evaluation, covering linear, probabilistic, ensemble, and neural network-based learning approaches. These models were chosen to represent different algorithm families, allowing a comprehensive performance comparison.

2.5. Training and evaluation

The dataset was divided into training (70%) and testing (30%) sets. To enhance reliability, five-fold cross-validation was applied. The performance of the classifiers was evaluated using five standard metrics: accuracy, precision, recall, F1-score, and false positive rate (FPR), following best practices in malware detection studies as reported by Yerima [13].

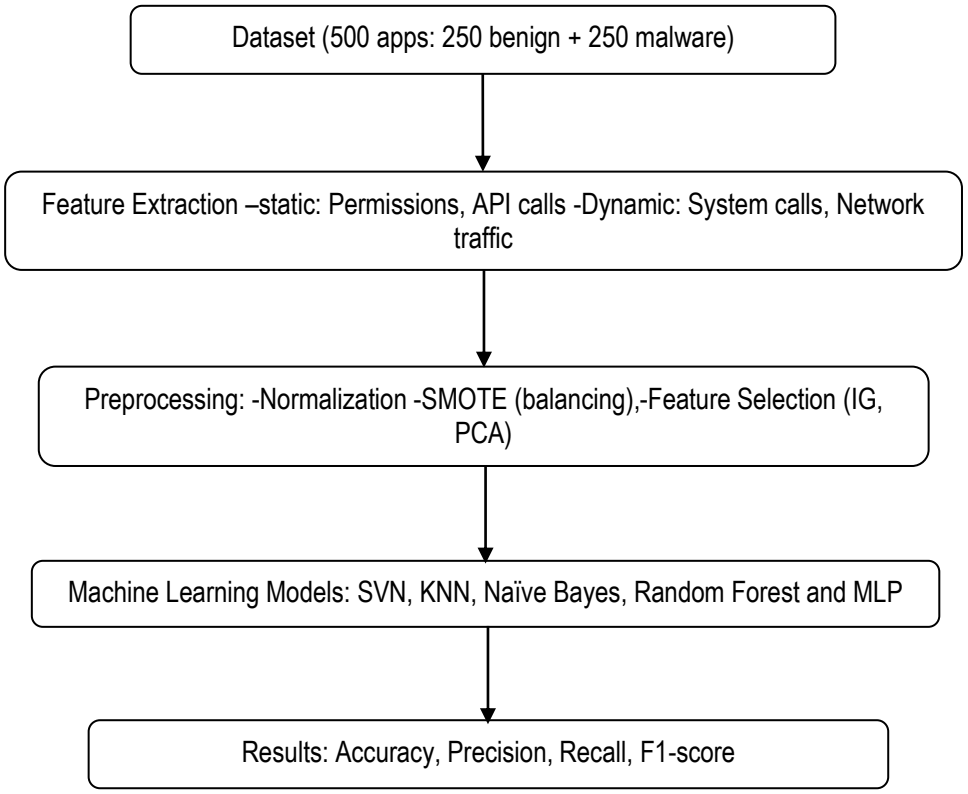

Figure 1: Proposed Android Malware Detection Framework

These metrics provide a balanced understanding of each model's predictive ability and reliability. Tables 2, 3 and 4 present the results of the classifiers based on these metrics.

3. Results and discussion

The performance of the classifiers was evaluated using the above listed metrics. These metrics provide a balanced assessment of the model's detection capability and reliability in distinguishing malware from benign.

3.1. Static Features

When using only static features (permissions, API calls, intent filters), the classifiers achieved moderate detection performance. KNN outperformed other models with an accuracy of 55.33% and F1-score of 56.21% as shown in Table 2. These findings highlight the limited discriminative power of static feature alone as reported by Barrera [3] and Arp [11], due to code obfuscation and repacking techniques. This subsection presents the results obtained from static analysis, where detection relied on features such as permissions, intents, and code-level attributes extracted from Android application packages (APKs). The performance of the classifiers was evaluated using accuracy, precision, recall, F1-score, and false positive rate (FPR). These metrics provide insight into how well each algorithm distinguishes benign from malicious applications using only static characteristics.

Table 2: Performance of Machine Learning Algorithms Using Static Features

Model	Precision (%)	Recall (%)	F1-Score (%)	Accuracy (%)	FPR (%)
SVM	53.16	56.00	54.55	53.33	49.2
KNN	55.13	57.33	56.33	55.33	46.8
Naïve Bayes	53.16	56.00	54.55	53.33	49.2
Random Forest	46.48	44.00	45.21	46.67	50.8
MLP	46.91	50.67	48.72	46.67	57.2

As shown in Table 2, the performance of the classifiers was generally moderate. Among the model, KNN achieved the highest accuracy and F1-score, indicating its relative ability to capture patterns within permission-and-code-based attributes. In contrast, Random Forest produced the lowest false positive rate, suggesting fewer benign applications were misclassified as malware. These results reveal that while static analysis can identify some malicious traits, it remains limited against obfuscation and code-hiding techniques.

3.2. Dynamic Features

Dynamic features (system calls, network traffic, and resource usage) significantly improved performance across classifiers. Random Forest achieved the best result with an F1-score of 84.08%., demonstrating the effectiveness of ensemble learning on runtime behaviors. These results are consistent with that of Karbab [14], who showed that behavioral features improve robustness against advanced malware.

However, dynamic analysis requires controlled environments and higher computational resources, which may limit scalability. In this phase, the classifiers were trained and evaluated using dynamic features obtained during the execution of Android applications. These features include API calls, network activities, and system behaviors captured through runtime monitoring. The evaluation, summarized in Table 3, was conducted using the same five metrics to ensure consistency and comparability with the static analysis results.

Table 3: Performance of Machine Learning Algorithms Using Dynamic Features

Model	Precision (%)	Recall (%)	F1-Score (%)	Accuracy (%)	FPR (%)
SVM	60.41	92.13	72.97	61.95	65.5
KNN	74.24	76.88	75.54	72.24	28.3
Naïve Bayes	71.99	20.24	31.59	51.13	8.8
Random Forest	83.42	84.76	84.08	82.11	17.5
MLP	53.46	70.56	60.55	52.80	67.1

Table 3 presents the performance of the classifiers based on dynamic features. The Random Forest algorithm achieved the best overall performance across all metrics, including the lowest false positive rate, confirming the effectiveness of behavioral features such as API calls and network activities. KNN and MLP also performed competitively, whereas Naïve Bayes showed weaker recall, indicating difficulty in modeling runtime variability. The superior results demonstrate that dynamic analysis provides richer behavioral information for accurate malware detection.

3.3. Combined static–dynamic features

Combining static and dynamic features further improved robustness. The integration allowed classifiers to learn from both code-level and behavioral attributes. Random Forest once again performed best, with the highest accuracy of 67.93% and an F1-score of 68.53% as shown in Table 4.

Although the improvement was not as high as in the purely dynamic case, the hybrid model reduced false positives, indicating better generalization. This finding aligns with Liu, [6], who demonstrated that combined approaches provide a balance between efficiency and accuracy. By integrating structural and behavioral information, the models were expected to achieve improved detection accuracy and lower false positive rates. The results, presented in Table 4, show how this combined feature set influences the overall performance of each classifier.

Table 4: Performance of Machine Learning Algorithms Using Combined Static and Dynamic Features

Model	Precision (%)	Recall (%)	F1-Score (%)	Accuracy (%)	FPR (%)
SVM	57.51	77.68	65.60	58.50	59.3
KNN	66.60	69.06	67.81	65.48	35.9
Naïve Bayes	64.46	34.54	40.77	52.01	21.6
Random Forest	68.64	68.46	68.53	67.93	31.7
MLP	53.46	70.56	60.55	52.80	63.7

The results of the combined static–dynamic feature set are summarized in Table 4. The Random Forest classifier again achieved the highest accuracy and F1-score while maintaining the lowest false positive rate, highlighting its robustness when handling hybrid feature inputs. The integration of static and dynamic attributes improved overall model balance, reducing false positives and capturing both structural and behavioral signatures. This finding supports the effectiveness of a hybrid approach for reliable Android malware detection.

3.4. Comparative Analysis

The comparative analysis across static, dynamic, and combined static-dynamic feature sets in Table 5 and Figure 1 shows that performance varies significantly depending on the type of features used. For static features, KNN achieved the best performance with an F1-score of 56.21%, indicating that while certain static attributes can capture limited malware behavior, overall performance remains modest. This confirms earlier findings of Barrera [3] and Arp [11] that static approaches are constrained by obfuscation and repackaging techniques. Dynamic analysis yielded the best overall results, with Random Forest achieving an F1-score of 84.08%. This demonstrates the value of runtime behavioral monitoring, consistent with Karbab, [14], who highlighted the strength of dynamic features in detecting sophisticated malware. However, the high resource cost of dynamic analysis limits its practicality

for real-time deployment. The combined static-dynamic approach balanced these trade-offs by integrating both static and dynamic features. While the F1-scores were slightly lower than those of the dynamic features alone, the combined static-dynamic models reduced false positives and improved generalization. This aligns with Liu, [6], who emphasized that combining static and dynamic features achieves a balance between computational efficiency and detection accuracy. Across classifiers, Random Forest consistently outperformed others in dynamic and combined settings, validating the effectiveness of ensemble methods for malware detection. In contrast, Naïve Bayes underperformed across all feature sets, reflecting its weakness in handling complex feature interactions.

Table 5: Performance comparison of classifiers across feature sets

Classifier	Static (F1-score %)	Dynamic (F1-score %)	Combined Static-dynamic (F1-score %)
SVM	54.55	72.97	65.60
KNN	56.33	75.54	67.81
Naive Bayes	54.55	31.59	40.77
Random Forest	45.21	84.08	68.53
MLP	48.72	68.44	60.55

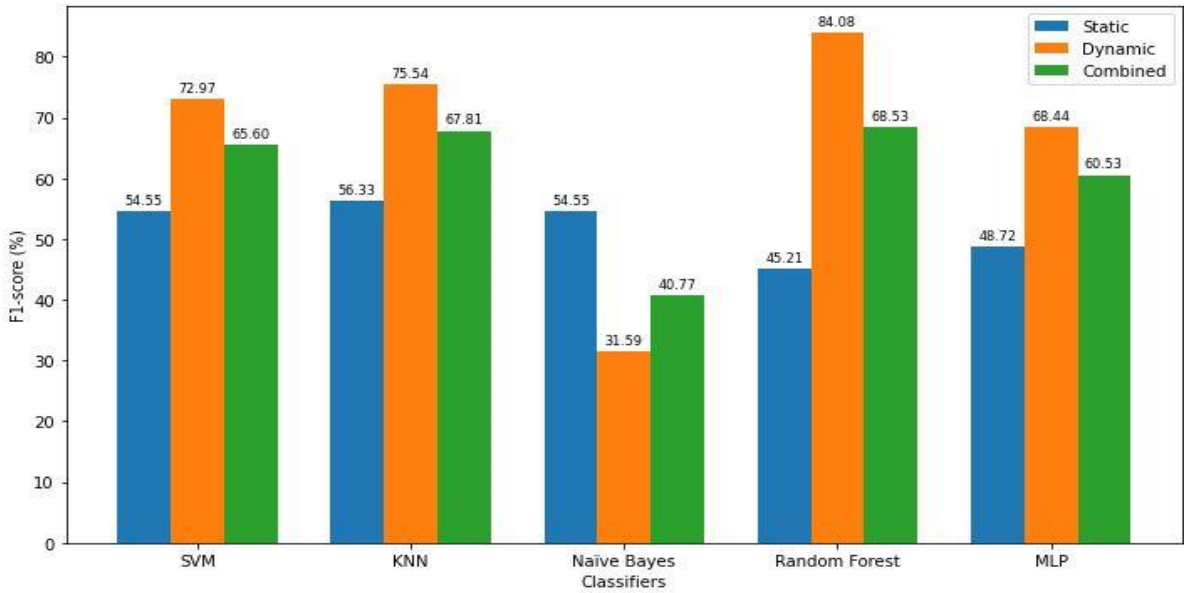

Figure 2:Performance Comparison of Classifiers Across Static, Dynamic and Combined Static-Dynamic Features

Overall, the results confirmed that dynamic analysis provides richer behavioral insights than static analysis, while combining static and dynamic features creates a more resilient framework. These findings align with prior research advocating static–dynamic integration for malware detection as reported by Liu [6] and Suarez-Tangil [1]. Ensemble learning methods, particularly Random Forest, demonstrated strong suitability for Android malware detection tasks.

Table 5 presents the F1-scores of the classifiers across static, dynamic and combined features. Figure 2 further illustrates these results, highlighting the superior performance of Random Forest in dynamic and combined settings.

As shown in Figure 2, Random Forest consistently outperformed other classifiers across all feature sets, confirming its robustness for Android malware detection.

4. Conclusion

The results demonstrated that static analysis alone provides modest accuracy, with KNN achieving the best F1-score of 56.33%. Dynamic analysis outperformed other approaches, with Random Forest achieving the highest F1-score of 84.08%. The combined static–dynamic approach provided balanced performance, with Random Forest again outperforming other classifiers and reducing false positives compared to individual feature sets. These findings confirm that integrating diverse features enhances robustness while highlighting Random Forest as the most reliable classifier in this context.

The contribution of this study lies in systematically evaluating multiple classifiers on integrated static and dynamic features, providing insights into the strengths and limitations of each approach. The results align with prior research while extending understanding through a unified experimental framework.

Future Work

While most detection frameworks focus on generic malware detection, category-specific models (e.g., SMS, banking malware) are increasingly important for real-world applications as reported by Manzil and Naik [15], showed that categorizing malware by type can improve detection relevance."

Future research should explore deep learning approaches to capture complex feature interactions, investigate larger datasets for improved generalization, and examine real-time implementation strategies for practical deployment in mobile security systems.

References

[1]. Suarez-Tangil, G., Tapiador, J., Peris-Lopez, P. and Ribagorda, A., (2013). Evolution, detection, and analysis of malware for smart devices, *IEEE Communications Surveys and Tutorials, 16*(2), 961–987. https://doi.org/10.1109/SURV.2013.111313.00198

[2]. Gandotra, E., Bansal, D., and Sofat, S. (2014). Malware analysis and classification: A survey, *Journal of Information Security, 5(2),* 56–64. https://doi.org/10.4236/jis.2014.52006

[3]. Barrera, D., Kayacik, H. G., van Oorschot, P. C. and Somayaji, A., (2010). A methodology for empirical analysis of permission-based security models and its application to Android, *In Proceedings of the 17th ACM Conference on Computer and Communications Security,* Chicago, IL, USA (pp. 73–84). ACM. https://doi.org/10.1145/1866307.1866317

[4]. Arora, A. and Peddoju, S. K., (2018). NTPDroid: A hybrid Android malware detector using network traffic and system permissions. *In 2018 IEEE 17th International Conference on Trust, Security and Privacy in Computing and Communications/12th IEEE International Conference on Big Data Science and Engineering (TrustCom/BigDataSE)* (pp. 808–813). IEEE. https://doi.org/10.1109/TrustCom/BigDataSE.2018.00115

[5]. Kabakus, A. T. and Dogru, I. A., (2018). An in-depth analysis of Android malware using hybrid techniques, *Digital Investigation, 24,* 25–33. https://doi.org/10.1016/j.diin.2018.01.013

[6]. Liu, Y., Guo, K., Huang, X., Zhou, Z. and Zhang, Y., (2018). Detecting Android malware with high-efficient combined static and dynamic feature analyzing methods, *Mobile Information Systems, 2018,* 1649703. https://doi.org/10.1155/2018/1649703

[7]. Haq, M. A. and Khuthaylah, M., (2024). Leveraging Machine Learning for Android Malware Analysis: Insights from Static and Dynamic Techniques, *Engineering, Technology and Applied Science Research*, 14(4), 15027–15032. https://doi.org/10.48084/etasr.7632

[8]. Alhussen, A., (2024). Advanced Android Malware Detection through Deep Learning Optimization, *Engineering, Technology and Applied Science Research*, 14(3), 14552–14557. https://doi.org/10.48084/etasr.7443

[9]. Reddy, K. S. U. Chakkaravarthy, S. S., Gopinath, M. and Aditya Mitra, A., (2023). A Study on Android Malware Detection Using Machine Learning Algorithms, ICISML 2022 (First EAI International Conference), 2023. https://doi.org/10.1007/978-3-031-35078-8_20.

[10]. Senanayake, J., Kalutarage H. and Al-Kadri, M. O., (2021). Android Mobile Malware Detection Using Machine Learning: A Systematic Review, *Electronics*, 10(13), Article 1606.

[11]. Arp, D., Spreitzenbarth, M., Hubner, M., Gascon, H., Rieck, K. and Siemens, C., (2014). Drebin: Effective and explainable detection of Android malware in your pocket. *In Proceedings of the 21st Annual Network and Distributed System Security Symposium (NDSS) (pp. 23-26), San Diego, CA, USA.* Internet Society. https://doi.org/10.14722/ndss.2014.23247

[12]. Alzaylaee, M. K., Yerima, S. Y. and Sezer, S., (2017). Evaluation of machine learning classifiers for Android malware detection, *Journal of Information Security and Applications, 34,* 28–39. https://doi.org/10.1016/j.jisa.2017.03.001

[13]. Yerima, S. Y., Sezer, S. and Muttik, I., (2014). Android malware detection using parallel machine learning classifiers, *In Proceedings of the 8th International Conference on Next Generation Mobile Applications, Services and Technologies (NGMAST),* Oxford, UK (pp. 37–42). IEEE. https://doi.org/10.1109/NGMAST.2014.11

[14]. Karbab, E. B., Debbabi, M., Derhab, A., and Mouheb, D. (2018). MalDozer: Automatic framework for Android malware detection using deep learning, *Digital Investigation, 24,* S48–S59. https://doi.org/10.1016/j.diin.2018.01.007

[15]. Manzil, H. R. and Naik, S. M., (2023). Android malware category detection using a novel feature vector-based machine learning model, *Cybersecurity*, 2023, Art. number (6:6).