Leveraging Machine Learning for Android Malware Analysis: Insights from Static and Dynamic Techniques

Mohd Anul Haq
Department of Computer Science, College of Computer Sciences and Information Sciences, Majmaah University; Al Majmaah, 11952; Saudi Arabia
m.anul@mu.edu.sa (corresponding author)

Majed Khuthaylah
Department of Information Technology, College of Computer Sciences and Information Sciences, Majmaah University, Al Majmaah; 11952; Saudi Arabia
441104924@s.mu.edu.sa

Received: 25 April 2024 | Revised: 4 May 2024 | Accepted: 17 May 2024

Licensed under a CC-BY 4.0 license | Copyright (c) by the authors | DOI: https://doi.org/10.48084/etasr.7632

ABSTRACT

In this study, the domain of Android malware detection was explored with a specific focus on leveraging the potential of Machine Learning (ML). At the time of this study, Android had firmly established its dominance in the mobile landscape and IoT devices, necessitating a concerted effort to fortify its security against emerging malware threats. Static analysis methods were scrutinized as vital sources of feature extraction for ML, while dynamic analysis methods were employed to analyze the behavior of applications in real or simulated environments. Additionally, a hybrid method, combining both static and dynamic analyses, was investigated. The study evaluated four ML models: XGBoost, Random Forest (RF), Support Vector Machine (SVM), and Decision Tree (DT), revealing compelling insights into their performance metrics. Notably, RF achieved the highest accuracy of 0.99, closely followed by SVM with an accuracy of 0.96. These results underscore the potential effectiveness of ML techniques in bolstering Android malware detection and mitigating security risks. As the research progressed, it underscored the latent power of integrating ML into the framework of Android malware analysis. With an eye towards the future, the overarching goal was to empower enhanced security measures and foster a resilient mobile ecosystem through the insights gleaned from this investigation.

Keywords-malware; ML; static and dynamic modeling; hyperparameter tuning; cross-validation; IoT

I. INTRODUCTION

Defending computer systems from digital threats is vital in today's Internet-driven world [1-4]. Traditional methods struggle against evolving attacks, making robust defense crucial. Intrusion detection, including firewalls and anti-virus software, is a key strategy. To enhance network protection, Machine Λearning (ML) plays a significant role. In this study, the focus is placed on using ML techniques, namely, XGBoost, RF, SVM, and DT, for malware detection. The novelty lies in applying cross-validation and rigorous hyperparameter tuning to these classifiers. This approach aims to improve their performance in identifying and preventing different types of malware, addressing the challenges posed by advanced cyber threats. Android, a widely employed mobile operating system, is an interesting target for malware due to its open-source nature and ability to install third-party applications without centralized control. The latest reports on malicious behaviors and vulnerabilities concentrate on the need for ongoing development to enhance Android's security. Due to the speed of technological developments, identifying malware application types has become even more important. Android malware has attracted grater attention during the last decade. Android OS is an extremely popular operating system, which dominates the market [5] with over 2.8 million available apps as of 2020 [6], most of them being available in Google Store. The increased number of users and application developers has made Android an attractive target for cyber-attacks [7]. This occurs because malicious programs easily execute their functions and are helped by some key elements, including the level of permissions, since some malware apps have over 50 variables, which makes detection difficult. On the contrary, the development of malware techniques has as its main goal to continue avoiding detection [9].

An experienced hacker also acting as constant attacker applies a tactic to remain alive by hiding from users as long as possible. Figure 1 describes how sophisticated malware attackers use an attack and exploitation plan. In the primary phase, which is reconnaissance and weaponization, the hackers look for weaknesses and vulnerabilities in the system, either on the hardware side, for example CPU, storage memory, chips, etc., or on the software side, e.g. zero-day vulnerabilities, not updating the system, weak encryption, etc. The next step entails preparing the threat and payload after finding the target to deliver the malware. Malware can be delivered either directly (e-mail, SMS, or links), or indirectly (encrypted links, directory attack, gains privilege access, adware, etc.). After the malware transfer, the next phase is exploitation, which leads to the activities applied by the malware. The following phase will possibly be shared between the active or passive strike. Access/bypass, hijacking/reply attack, DoS/DDoS are examples of active attacks. Monitoring network traffic, scanning (open ports/vulnerabilities), and spying are examples of passive attacks. After the second phase, usually, the attacker chooses between quitting or continuing. Even if the attacker quits, the threat continues to exist. The attacker tries to gain access to further information by installing malicious software and creating a backdoor. By stealing the credentials the attackers can be active as long as possible. The afterward phase is C&C (Command and Control), which allows the attacker to gain continual connection, this could infect other connected systems, which makes it act like/resemble Data Exfiltration, and spread laterally. Data exfiltration involves unauthorized transfer of data from a system, often to external servers or devices. It is a crucial step where sensitive information, such as personal or financial data, is stolen. Lateral spreading refers to the malware's ability to move within a network, infecting multiple devices or systems. These processes allow the malware to expand its reach, maximize damage, and achieve its objectives, posing significant threats to data security and system integrity. Therefore, in the continuous Android malware development, it is necessary to work on ways that deal with it [5-7].

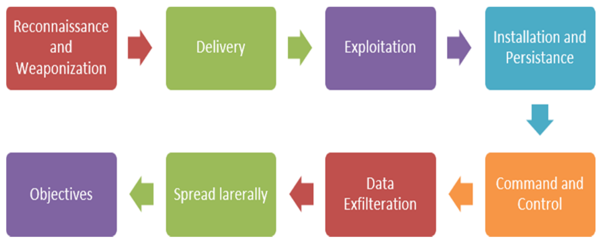

Fig. 1. Malware operation phase.

Three techniques have been developed for Android malware detection, which are static, dynamic, and hybrid. Static analysis can determine harmful applications by their behavior. The problem here is code obfuscation that helps the malware avoid the static technique. The dynamic technique helps identify the conduct of malware applications from runtime. Hybrid technique uses features of both static and dynamic techniques, including analyzing the application's source files and monitoring runtime behavior [9].

A recently utilized technology is ML. Extracting and selecting features are important procedures in identifying the malware process steps in the Android operating system. The adequacy in feature extraction increases system performance. So, paying more attention to the procedures for determining and digging out the features that are needed from the hotspot in the malicious detecting process is required [10-12]. In the feature selection process, some challenges could be faced: The former may take a long time because the Android data file has a lot of features and therfore a huge size of data, which could lead to multiple problems during the extracting features process. Because of this, there is a possibility that the detecting malware process will not be effective since the accuracy of the latter could be affected by the selection of some features. Extracting and selecting features is still a developing field. Most of the time, a feature can be determined to allow the algorithm to ignore executing program paths that are not important, thereby compromising accuracy performance. Hence, the primary objective of this research is to identify the most accurate feature for detecting malware using static, dynamic, and hybrid analysis. The article discusses proper classification by optimally selecting features.

To face malware attacks, security solutions using ML approaches, static analysis, and dynamic analysis, is widely considered. Data science is presented as a promising domain in cybersecurity, by leveraging analytical models to discover potential malicious activities. The model introduced in [13] engages permission features from the Android system for malware detection. A static malware analysis method was introduced in [14] for classification training to reveal whether the app is virus or friendly. The authors extracted the API call feature by adopting correlative analysis with utilizing three different ML models (SVM, RF, and KNN), with the RF exhibiting the best accuracy result. In [15], another approach for static analysis was introduced, which follows three steps to determine features. The authors found that RF had the best accuracy result among the considered classifiers. Authors in [16] leveraged the RF, which resulted in 96% accuracy in malware detection by operating static testing. Authors in [17] leveraged ML categorization to investigate the problem of defense techniques against malware. Authors in [18] achieved a precision of 96.9% with KNN and SVM. At the same time, other works have found distinct methodologies to categorize malware attacks using dynamic examination.

The present study delves into the domain of Android malware detection, emphasizing the growing importance of strengthening security measures against emerging threats. With Android's dominant presence in the mobile landscape and IoT devices, there is a pressing need to address vulnerabilities and fortify defenses. The investigation explores the potential of ML techniques in enhancing malware detection. By examining both static and dynamic analysis methods, the research aims to extract valuable insights into feature extraction and behavior analysis, crucial aspects of effective malware detection. Through this exploration, the study seeks to contribute to the advancement of security measures in the Android ecosystem, laying the groundwork for resilient defenses against evolving malware threats.

II. METHODOLOGY

A variety of Python libraries were incorporated into the research work for an effective ML analysis. The scikit-learn library provided essential tools for implementing various classifiers, such as XGBoost, RF, SVM, and DT. These classifiers helped build, train, and evaluate models for specific needs, involving malware prediction. Additionally, Pandas was used for data manipulation and analysis, while Pumpy supported numerical operations. The Seaborn Library facilitated the creation of visually appealing plots and charts for better data representation. Other libraries, like Joblib and Pickle aided in saving and loading models, ensuring efficient model management. The code utilizes functionalities like label encoding, robust scaling, train-test split, and metrics for model evaluation. It employs GridSearchCV for hyperparameter tuning, confusion matrix visualization, and Receiver Operating Characteristic (ROC) curve plotting. The overall manuscript flow is depicted in Figure 2.

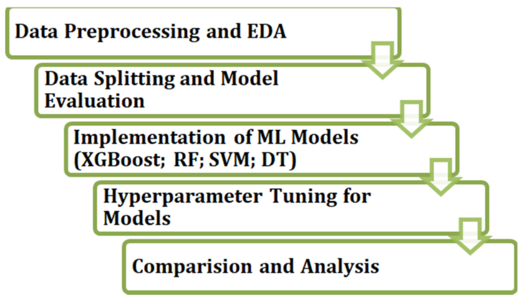
Data Preprocessing and EDA
Data Splitting and Model Evaluation
Implementation of ML Models (XGBoost; RF; SVM; DT)
Hyperparameter Tuning for Models
Comparision and Analysis

Fig. 2. Overall workflow of the present investigation for malware detection.

A. *Dataset Pre-Processing and Exploratory Data Analysis*

The present investigation utilized a genuine malware dataset [20]. Originating from studies in Android security employing ML, this dataset underwent thorough data preprocessing. The process involved generating binary vectors to signify permissions deployed in each analyzed application (1 for used, 0 for not used). Additionally, models were categorized by type, distinguishing between malicious (1) and non-malicious (0) applications. The dataset encompasses 398 records across 331 columns [20]. To identify crucial features for malware detection, the analysis prioritized extracting the top 10 permissions commonly utilized by both malicious and non-malicious entities, highlighting the importance of preprocessing in preparing the data for effective analysis.

B. *ML Models*

The present inevstigation used 4 ML models to analyze and enhance malware detection.

1) *XGBoost*

GBoost, a specialized Decision Tree (DT), was employed to achieve fast and superior performance. This tree follows techniques such as gradient boosting and multiple regression trees. The concept of boosting involves training new models based on older ones, and progressively refining them until further improvements are minimal. The well-known AdaBoost method within XGBoost assigns weights to data points, proving valuable for predictions.

2) *RF*

Random Forest (RF) extends the concept of DTs by creating a group of DTs based on samples selected from a larger dataset. Unlike a single DT, RF introduces feature randomness by considering only a subset of features, preventing them from being too similar. The final prediction in RF is determined by the majority vote from all DTs. This approach enhances accuracy and reliability compared to relying on a single DT.

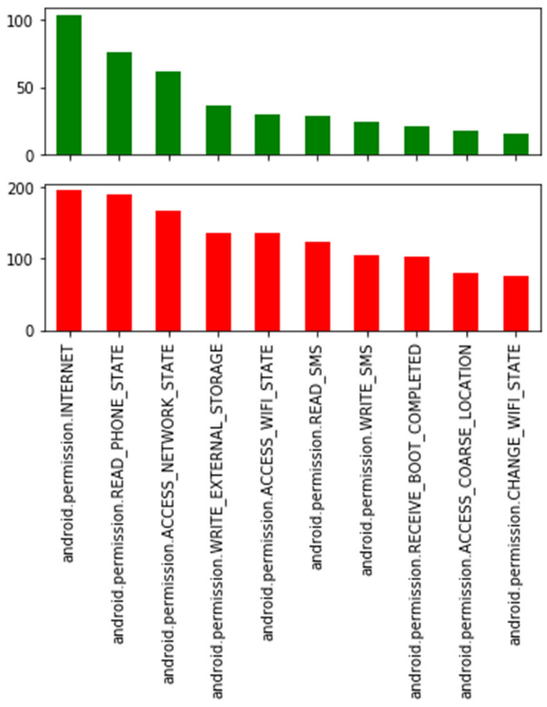

Fig. 3. Permissions for friendly (green) and malicious (red).

3) *Support Vector Machine*

Support Vector Machine (SVM) emerged as a robust method for improving model accuracy while avoiding overfitting. SVM can classify data points even if they are not linearly distinct. The SVM transforms data into a hyperplane representation after identifying a partition between classes.

4) *Decision Tree*

Decision Tree (DT) is a learning method used for categorization and regression. It is a hierarchical tree that includes a basis node, subdivisions, inner nodes, and leaves. DT learning utilizes a rapacious search method to find the best-divided points, employing a divide-and-conquer strategy. The procedure is repeated recursively until most records are assigned to specific class labels.

C. *Hyperparameter Tuning*

Hyperparameter tuning is essential because it optimizes the performance of ML models by fine-tuning the parameters that are not directly learned from the data. This process helps to find the best combination of hyperparameters, leading to improved model accuracy and generalization to unseen data. To ensure the optimal performance of the models without overfitting the training data, a method called Gridsearchcv was employed. Gridsearchcv was chosen over other methods for its ability to ensure the robustness of the models and avoid an excessive focus on the training data. This facilitated the adjustment of crucial settings within the models, such as the regularization parameter (C), kernel type, and kernel coefficient (gamma) for SVM. Similar adjustments were made for parameters like the number of estimators, learning rate, and maximum depth for XGBoost.

TABLE I. FINE-TUNED HYPERPARAMETERS OF ALL MODELS

Model	Hyperparameter	Tuned value
XGBoost	learning rate: [0.01, 0.1, 0.2]	0.01
	N estimators: [50, 100, 200]	100
	Max depth: [3, 5, 7]	3
	Min child weight: [1, 3, 5]	
RF	N estimators: [50, 100, 200]	100
	Max depth: [None, 5, 10]	10
	Min samples split: [2, 5, 10]	5
	Min samples leaf: [1, 2, 4]	4
SVM	C: [0.1, 1, 10]	1
	kernel: [linear, rbf, poly,]	linear
	gamma: [scale, auto, 0.1, 1]	auto
DT	Max depth: [None, 5, 10]	5
	Min samples split: [2, 5, 10]	5
	Min samples leaf: [1, 2, 4]	2
	criterion: [gini, entropy]	gini

D. Cross Validation

To ensure robustness and reliability in assessing the models' performance, the 5-fold cross-validation approach was adopted. Cross-validation is a crucial technique for training and evaluating models using multiple subsets of the dataset. In the case of this investigation, the dataset was divided into five folds, and the modeling process was iteratively performed 5 times. Through every repetition, one fold was implemented for the validation set, whereas the residual folds were utilized for exercising the simulation. This procedure was repeated five times, through every fold selecting a turn as the validation set. The benefits of using 5-fold cross-validation are manifold. It reduces the risk of overfitting, granting an additional tough estimate of the simulation's functioning. Additionally, it contributes to a greater accurate understanding of how the model is possible to generalize to hidden data.

E. Evaluation Measures

The ML models' performance was assessed with four measures: recall, F1-score, precision, and accuracy. Accuracy reflects the proportion of correctly recognized spam emails among all regular emails. Recall is the proportion of correctly recognized spam messages to the total real spam samples, whereas accuracy is the percentage of correctly recognized spam messages to the entire messages identified as spam. F1-score is deployed to assess the effectiveness of the approach.

$$\text{Accuracy}=\frac{\text{TP+TN}}{\text{TP+FP+TN+FN}} \qquad (1)$$

$$\text{Precision}=\frac{\text{TP}}{\text{TP+FP}} \qquad (2)$$

$$\text{Accuracy}=\frac{\text{TP}}{\text{TP+FN}} \qquad (3)$$

$$\text{F1-score}=\frac{\text{Precision+Recall}}{\text{Precision}\times\text{Recall}} \qquad (4)$$

where TN = true negative, TP = true positive, FP = false positive, and FN = false negative.

Two metrics, the Receiver runs the Characteristic (ROC) curve and Area Under the Curve (AUC), were examined. The ROC curve illustrates the model's ability to distinguish between attack and non-attack by plotting the comparison between TP and FP rates. A greater AUC indicates better overall performance, measuring the field below the ROC curve. These metrics afford a clear vision keen on the model's effectiveness in classifying security attacks.

III. RESULTS AND DISCUSSIONS

Android malware detection was assessed with four ML models. RF exhibited the highest accuracy of 99%, with perfect precision and recall for detecting malicious apps (Class 1). XGBoost closely followed with an accuracy of 94%, maintaining balanced precision and recall for both classes. SVM also performed well, achieving an accuracy of 96% with consistent precision and recall scores. DT showed a slightly lower accuracy of 95%, with slightly imbalanced precision and recall for both classes (Table II).

TABLE II. PERFORMANCE EVALUATION

Model	Accuracy	Class 0			Class 1		
		Precision	Recall	F1-Score	Precision	Recall	F1-Score
XGBoost	0.94	0.94	0.93	0.94	0.94	0.94	0.94
RF	0.99	0.99	1.00	0.99	1.00	0.98	0.99
SVM	0.96	0.96	0.96	0.96	0.96	0.96	0.96
DT	0.95	0.92	0.98	0.95	0.98	0.91	0.95

The confusion matrix (Figure 4) helps analyze the ML models for malware finding by applying four measures to demonstrate the results of (F1-score, recall, precision, and accuracy). In Figure 4, 1 means positive, and 0 is negative. The ROC curve illustrates a binary classifier's performance, showcasing the trade-off between true positive and false positive rates (Figure 5). The AUC summarizes this performance, with higher values indicating better classifier performance.

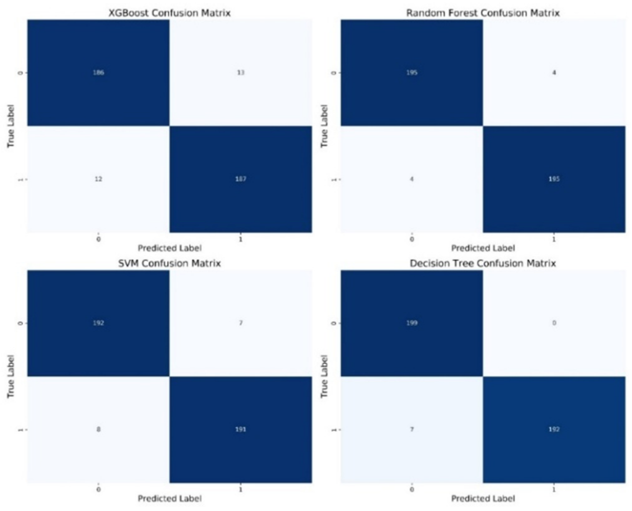

Fig. 4. Confusion matrix of all 4 ML malware detection models.

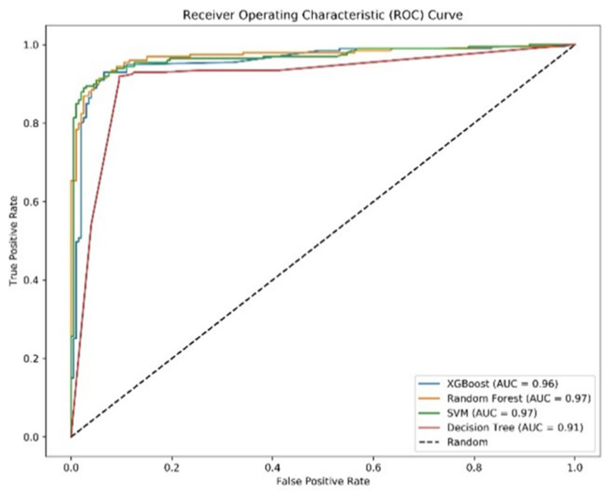

Fig. 5. ROC curves with AUC values for all 4 ML models.

IV. COMPARISON WITH OTHER WORKS

Various studies have explored Android malware detection using different features and classification algorithms. Authors in [19] utilized permissions and API calls, achieving a classification performance of 98% employing RF. Authors in [21] focused on permissions alone, attaining 92% accuracy deploying RF.

TABLE III. COMPARISON WITH OTHER STUDIES

Ref	Feature used	Classification algorithm	Classification performance
[19]	Permissions, API call	RF	0.98 (f-measure)
[21]	Permissions	RF	92 (accuracy%)
[9]	Permissions	SVM	90 (accuracy%)
This work	Permissions	XGBoost	94 (accuracy%)
	Permissions	RF	99 (accuracy%)
	Permissions	SVM	96 (accuracy%)
	Permissions	DT	95 (accuracy%)

Similarly, authors in [9] reached 90% accuracy by employing permissions with SVM. In comparison, the present study concentrated on permissions and engaged XGBoost, RF, SVM, and DT. RF demonstrated the highest accuracy at 99%, followed by SVM at 96%, XGBoost at 94%, and DT at 95%.

V. STRENGTHS AND LIMITATIONS OF THE CURRENT INVESTIGATION

The study's strength lies in its focused exploration of extracting the top 10 permissions using diverse ML techniques, aiming to identify malicious findings for optimal results. However, it is essential to acknowledge potential limitations, such as data availability, model complexity, and the evolving nature of malware, which may impact the comprehensiveness and generalizability of the findings.

VI. CONCLUSION

This study highlighted the importance of employing Machine Learning (ML) algorithms for Android malware detection. Features such as permissions play a crucial role in classification accuracy. RF emerged as a particularly effective algorithm, consistently achieving high accuracy in the current study and in similar works. Additionally, the discussion performed emphasized the significance of meticulous data preprocessing in enhancing the performance of ML models. These findings underscore the potential of ML in bolstering Android security measures and mitigating the risks posed by malicious applications.The future scope of the present investigation is to utilize advanced malware detection models [22-24] and validate them for various applications [25-28].

ACKNOWLEDGMENT

Mohd Anul Haq would like to thank the Deanship of Postgraduate Studies and Scientific Research at Majmaah University for supporting this work under Project No. R-2024-1120.

REFERENCES

[1] M. A. Haq, "DBoTPM: A Deep Neural Network-Based Botnet Prediction Model," *Electronics*, vol. 12, no. 5, Jan. 2023, Art. no. 1159, https://doi.org/10.3390/electronics12051159.

[2] M. A. Haq and M. A. R. Khan, "Dnnbot: Deep neural network-based botnet detection and classification," *Computers, Materials and Continua*, vol. 71, no. 1, pp. 1729–1750, 2022, https://doi.org/10.32604/cmc.2022.020938.

[3] M. A. Haq, M. A. R. Khan, and T. AL-Harbi, "Development of pccnn-based network intrusion detection system for edge computing," *Computers, Materials and Continua*, vol. 71, no. 1, pp. 1769–1788, 2022, https://doi.org/10.32604/cmc.2022.018708.

[4] C. S. Yadav *et al.*, "Malware Analysis in IoT & Android Systems with Defensive Mechanism," *Electronics*, vol. 11, no. 15, Jan. 2022, Art. no. 2354, https://doi.org/10.3390/electronics11152354.

[5] H. Cai, X. Fu, and A. Hamou-Lhadj, "A study of run-time behavioral evolution of benign versus malicious apps in android," *Information and Software Technology*, vol. 122, Jun. 2020, Art. no. 106291, https://doi.org/10.1016/j.infsof.2020.106291.

[6] H. Cai and B. Ryder, "A Longitudinal Study of Application Structure and Behaviors in Android," *IEEE Transactions on Software Engineering*, vol. 47, no. 12, pp. 2934–2955, Sep. 2021, https://doi.org/10.1109/TSE.2020.2975176.

[7] M. Noman and M. Iqbal, "A Survey on Detection and Prevention of Web Vulnerabilities," *International Journal of Advanced Computer Science and Applications*, vol. 11, no. 6, pp. 521–540, Jul. 2020, https://doi.org/10.14569/IJACSA.2020.0110665.

[8] A. S. Shatnawi, A. Jaradat, T. B. Yaseen, E. Taqieddin, M. Al-Ayyoub, and D. Mustafa, "An Android Malware Detection Leveraging Machine Learning," *Wireless Communications and Mobile Computing*, vol. 2022, May 2022, Art. no. e1830201, https://doi.org/10.1155/2022/1830201.

[9] K. Liu, S. Xu, G. Xu, M. Zhang, D. Sun, and H. Liu, "A Review of Android Malware Detection Approaches Based on Machine Learning," *IEEE Access*, vol. 8, pp. 124579–124607, 2020, https://doi.org/10.1109/ACCESS.2020.3006143.

[10] W. Zhang, H. Wang, H. He, and P. Liu, "DAMBA: Detecting Android Malware by ORGB Analysis," *IEEE Transactions on Reliability*, vol. 69, no. 1, pp. 55–69, Mar. 2020, https://doi.org/10.1109/TR.2019.2924677.

[11] S. Alam, S. A. Alharbi, and S. Yildirim, "Mining nested flow of dominant APIs for detecting android malware," *Computer Networks*, vol. 167, Feb. 2020, Art. no. 107026, https://doi.org/10.1016/j.comnet.2019.107026.

[12] O. Olukoya, L. Mackenzie, and I. Omoronyia, "Towards using unstructured user input request for malware detection," *Computers & Security*, vol. 93, Jun. 2020, Art. no. 101783, https://doi.org/10.1016/j.cose.2020.101783.

[13] N. J. Ratyal, M. Khadam, and M. Aleem, "On the Evaluation of the Machine Learning Based Hybrid Approach for Android Malware Detection," in *22nd International Multitopic Conference*, Islamabad, Pakistan, Nov. 2019, pp. 1–8, https://doi.org/10.1109/INMIC48123.2019.9022790.

[14] H. Zhang, S. Luo, Y. Zhang, and L. Pan, "An Efficient Android Malware Detection System Based on Method-Level Behavioral Semantic Analysis," *IEEE Access*, vol. 7, pp. 69246–69256, Jan. 2019, https://doi.org/10.1109/ACCESS.2019.2919796.

[15] C. Yang, Z. Xu, G. Gu, V. Yegneswaran, and P. Porras, "DroidMiner: Automated Mining and Characterization of Fine-grained Malicious Behaviors in Android Applications," in *19th European Symposium on Research in Computer Security*, Wroclaw, Poland, Sep. 2014, vol. 8712, pp. 163–182, https://doi.org/10.1007/978-3-319-11203-9_10.

[16] H. Fereidooni, M. Conti, D. Yao, and A. Sperduti, "ANASTASIA: ANdroid mAlware detection using STatic analySIs of Applications," in *8th IFIP International Conference on New Technologies, Mobility and Security*, Larnaca, Cyprus, Nov. 2016, pp. 1–5, https://doi.org/10.1109/NTMS.2016.7792435.

[17] X. Fu and H. Cai, "On the Deterioration of Learning-Based Malware Detectors for Android," in *41st International Conference on Software Engineering: Companion Proceedings*, Montreal, QC, Canada, Dec. 2019, pp. 272–273, https://doi.org/10.1109/ICSE-Companion.2019.00110.

[18] L. Cai, Y. Li, and Z. Xiong, "JOWMDroid: Android malware detection based on feature weighting with joint optimization of weight-mapping and classifier parameters," *Computers & Security*, vol. 100, Jan. 2021, Art. no. 102086, https://doi.org/10.1016/j.cose.2020.102086.

[19] R. S. Arslan, I. A. Dogru, and N. Barisci, "Permission-Based Malware Detection System for Android Using Machine Learning Techniques," *International Journal of Software Engineering and Knowledge Engineering*, vol. 29, no. 01, pp. 43–61, Jan. 2019, https://doi.org/10.1142/S0218194019500037.

[20] C. Urcuqui, "Dataset malware/beningn permissions Android." 2016, [Online]. Available: https://www.kaggle.com/datasets/xwolf12/datasetandroidpermissions.

[21] G. Tao, Z. Zheng, Z. Guo, and M. R. Lyu, "MalPat: Mining Patterns of Malicious and Benign Android Apps via Permission-Related APIs," *IEEE Transactions on Reliability*, vol. 67, no. 1, pp. 355–369, Mar. 2018, https://doi.org/10.1109/TR.2017.2778147.

[22] A. Al-Marghilani, "Comprehensive Analysis of IoT Malware Evasion Techniques," *Engineering, Technology & Applied Science Research*, vol. 11, no. 4, pp. 7495–7500, Aug. 2021, https://doi.org/10.48084/etasr.4296.

[23] K. Aldriwish, "A Deep Learning Approach for Malware and Software Piracy Threat Detection," *Engineering, Technology & Applied Science Research*, vol. 11, no. 6, pp. 7757–7762, Dec. 2021, https://doi.org/10.48084/etasr.4412.

[24] A. Bathula, S. Muhuri, S. Gupta, and S. Merugu, "Secure certificate sharing based on Blockchain framework for online education," *Multimedia Tools and Applications*, vol. 82, no. 11, pp. 16479–16500, May 2023, https://doi.org/10.1007/s11042-022-14126-x.

[25] A. Bathula, S. Gupta, S. Merugu, and S. S. Skandha, "Academic Projects on Certification Management Using Blockchain- A Review," in *International Conference on Recent Trends in Microelectronics, Automation, Computing and Communications Systems*, Hyderabad, India, Dec. 2022, pp. 1–6, https://doi.org/10.1109/ICMACC54824.2022.10093679.

[26] S. Merugu, K. Jain, A. Mittal, and B. Raman, "Sub-scene Target Detection and Recognition Using Deep Learning Convolution Neural Networks," in *ICDSMLA 2019*, Singapore, 2020, pp. 1082–1101, https://doi.org/10.1007/978-981-15-1420-3_119.

[27] M. Suresh, A. S. Shaik, B. Premalatha, V. A. Narayana, and G. Ghinea, "Intelligent & Smart Navigation System for Visually Impaired Friends," in *12th International Advanced Computing Conference*, Hyderabad, India, Dec. 2022, pp. 374–383, https://doi.org/10.1007/978-3-031-35641-4_30.

[28] S. Merugu, M. C. S. Reddy, E. Goyal, and L. Piplani, "Text Message Classification Using Supervised Machine Learning Algorithms," in *International Conference on Communications and Cyber Physical Engineering*, Hyderabad, India, Jan. 2018, pp. 141–150, https://doi.org/10.1007/978-981-13-0212-1_15.