Adopting Graph-Based Machine Learning Algorithms to Classify Android Malware

Abdelrahman Elsharif Karrar
College of Computer Science and Engineering
Taibah University, Medina, Saudi Arabia

Abstract
As mobile device usage grows, it is worth noting that smartphones are among the most important inventions of the century. The evolution of smartphones and access to affordable internet has made technology an integral part of our daily lives. Android operating systems have provided an adaptable environment for hackers to develop new mobile applications loaded with malware through which attacks such as denial of service and privacy breaches are executed. Malware developers exploit vulnerabilities in the installation and runtime files to execute cyberattacks on the devices. The present study adopts a graph-based machine learning algorithm to manage imperative permissions and API functionalities using application data from the Drebin project, in which 15,036 applications were tested to determine the most important features for malware detection. Machine learning techniques such as Logistic Regression Algorithm (LR), Decision Tree Algorithm (DT), K-Nearest Neighbor Algorithm (KNN), and Random Forest (RF) Algorithm are used in the classification and training of malware detection programs. The findings suggest that the RF technique achieves the highest rate of recall (96%) and accuracy (97%) while KNN and DT deliver (96%) accuracy while LR delivers (95%).

Keywords:
Graph-Based Model; Machine Learning; Classification Algorithms; Android Malware Detection.

1. INTRODUCTION

Smartphones are the most popular mobile devices used to perform a wide range of functions to aid users' day-to-day activities. The devices have created a new social connectivity aspect, especially due to the rapid adoption in military systems, enterprises, and state agencies[1]. Smartphones expose users to potential cybersecurity threats caused by black market applications freely accessed by unregulated developers. Successful exploitation of Android vulnerabilities by black market applications could potentially limit user activity and modify or transmit sensitive data without authorization[2].

It is noticeable that the Android operating system is the most popularly used in smartphones. Also, the market for Android applications is largely unregulated, creating a potential risk of malicious applications being downloaded into user devices. Due to the rapid proliferation of freely available Android applications, users are exposed to a significant risk of undetected malware.

Therefore, mobile app developers must introduce novel configurations to guarantee the integrity and privacy of user information[3] using countermeasures such as signature-based antivirus scanners and malware classification programs[4][5]. Such countermeasures deliver varying rates of accuracy depending on their functional mechanisms.

Machine learning approaches have been broadly utilized in detecting malware through classic pattern recognition based on static and dynamic mechanisms to ensure scalability to cybersecurity's changing scope [6]. Machine learning techniques outperform traditional static and dynamic time consumption and scalability approaches since they thoroughly explore the substances and software details[7]. Feature selection and learning rate are the main factors that influence the performance of ML-based cybersecurity solutions due to their superior performance in detecting false positives[8].

This study aims to develop a graph-based model for classifications of Android malware utilizing machine learning strategies and detect unidentified malware by utilizing machine learning techniques with low resource requirements and computation costs by exploring the mobile device security from the context of API calls and sensitive permissions, which are integrated, trained, and tested by embedding classifier models in feature vectors space[1]. This paper is focused on adopting an adaptable feature security approach to deliver superior feature generation performance with reduced resource consumption.

The rest of this paper is structured as follows: Section 2 reviews related work. Section 3 demonstrates the proposed methodology. Section 4 evaluates the experiments and discusses the results. Finally, Section 5 concludes the paper by identifying limitations and recommendations for future work.

Manuscript received September 5, 2022
Manuscript revised September 20, 2022
https://doi.org/10.22937/IJCSNS.2022.22.9.109

2. RELATED WORK

2.1. Mobile Android systems and Mobile Android apps

Android started as a project of the U.S technology corporation Android Inc. in 2003 to design OS for a digital camera [9]. In November 2007, the present Android version was called "Beta" as the original formal commercial version. The Android version has been coded under Sweet since April 2009 and distributed alphabetically[10]. Android operating system is established on the Linux kernel and operates for cellular computers and phones[4]. Typically, the Android phone comes with various in-built apps. Also, it supports third-party software and apps[11]. Designers can develop software for Androids using the free SDK (Android software designer kits)[12]. Android software is published in Java and runs via Java virtual machines (JVMs) that are improved for mobile gadgets.

Also known as mobile apps or just apps, mobile applications care software applications or computer programs developed to operate mobile gadgets like watches, phones, and tablets[13]. Mobile applications are downloaded from app distribution networks run by OS owners such as Google Play Store or APP Store (iOS), who provide certain free applications and others at a fee.

2.2. Android security architectures

Android architectures make applications of isolation models to design operating systems with security features [12]. As illustrated in Fig. 1, the Android operating systems are developed on top of the Linux kernel. The Linux kernels are responsible for integrating primary system services like physical device access through network management, driver, power management, process management, and memory accessibility. Libraries and the Linux kernel are the major building blocks of Android platforms. Linux kernel, the bottom Android architecture layer, offers fundamental model functionality such as device management like system drivers, Bluetooth, displays, keypads, cameras, and process management[14]. Libraries, the top Android architecture consists of SSL libraries accountable for internet security, libraries for playing and recording audios and videos, SQLite database for application data sharing and storage, Free Type, SGL, media frameworks, well-known labs, and open-source web browser engines and web-kits.

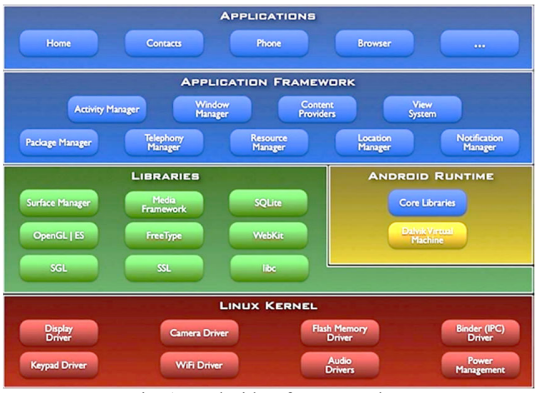

Fig. 1. Android Software Stacks

Android runtime is the third part in the second section from the bottom in Android architecture. This part offers major components known as Dalvik Virtual Machines, particularly developed and optimized for Androids [15]. The application framework section offers multiple high-level services to apps in the form of java programs. The app framework directly interacts with the Android architecture blocks [12].

2.3. The Android attack surfaces

Attack surfaces are the target features that make android applications susceptible to intrusions or attacks. Attack vectors refer to the ways by which intruders exploit vulnerabilities to engage in malicious activities[9]. Contrary to attack vectors, attack surfaces do not rely on intruder activities or existing vulnerabilities as they are often undiscovered code susceptibilities. The system can be secured or exploited more rapidly by focusing on specific attack surfaces. Therefore, it is essential to categorize android devices because they have such complex attack surfaces, which may be difficult to detect using standard techniques [10]. Some of the more basic attack surfaces for android gadgets, along with certain prorogation mechanisms and attack vectors, are exemplified in Fig. 2;

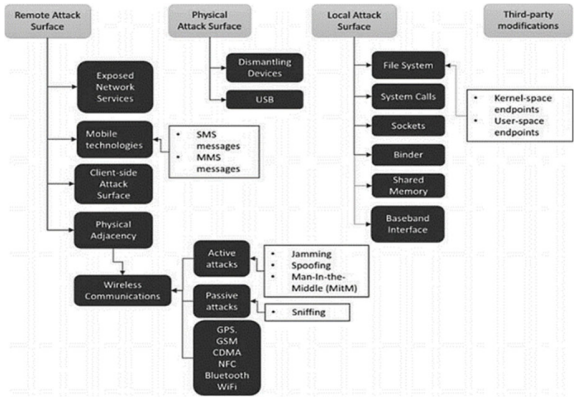

Fig. 2. Android Attack Surfaces

2.4. Mobile Malware definition and Malware life cycle

Mobile malware is malicious software or programs that are particularly developed to target phone gadgets like tablets and smartphones to get accessibility to confidential information [16].

Malware for phone devices in general and Androids in specific replicate the virus behaviors experienced on desktops [17]. Their life cycle is designed in seven major stages.

In the creation stage, the programmers design and execute all malicious codes that will be integrated into the malware.

In the Gestation stage, the malicious apps infiltrate and settle in the systems they want to infect [18]. Malicious apps remain inactive throughout this phase, so their presences remain completely unknown to the users.

The infection or reproduction stage is where the malware reproduces many times before manifestation [9]. The malware authors seek to access confidential information by remotely controlling gadgets. The malware spreads through social engineering or file-sharing approaches on Androids [19]. They use Wi-Fi, SMS, and Bluetooth as communication techniques and usually disguise themselves as ordinary applications. Inactivation stage, some malware activates their damage routines when specific conditions are met [12]. The users notice strange behaviors and suspect the presence of malicious apps in the discovery phase.

The strange behaviors can include the unavailability of some system functions, performance losses, and current changes in the homepage of a web browser [20]. Antivirus updates its virus databases after new malware discovery in the assimilation stage.

Antidotes or fixes are also recommended -if possible- to remove these threats[21]. The elimination phase is when the antiviruses discover the malware and prompt the users to eliminate them [18]. It marks the malware's death.

2.5. Mobile Malware types

The various malware intrusions include Trojans, spyware, ransomware, mobile phishing attacks, mobile bots, worms, and viruses. Computer worms are malware types that infect the device and remain active on the infected system. Mobile bots are malware types that automatically run after users install them on their devices [13]. They gain full access to the devices and their contents and start getting instructions and communicating with one or more control servers and commands [22].

Mobile phishing attacks usually come in SMS or email text messages. The attacks masquerade reputable entities or people and distribute malicious attachments and links that can account for data from a victim or extract login information.

Ransomware is malware that locks the information on victims' devices and the devices themselves, typically through encryption. The intruders then demand payments before the devices, or data ace are returned to the victims or decrypted [23].

Spywares synchronize with personal data sources such as notes, email accounts, passwords, and calendar applications and gather that information and data and send them to remote servers.

Trojan horse viruses require users to activate them[24]. Intruders insert Trojan into non-malicious executable apps or files on mobile devices.

2.6. Android Malware Detection Tools

Antivirus software, firewalls, and intrusion detection system (IDS) are the three tools used for malware discovery [25]. An intrusion detection system (IDS) monitors the networks for policy violations and malicious activities. Antivirus designers initially used the attack signatures to scan system files for malicious activity evidence. Signature-oriented detection systems monitor inbound network traffics for patterns and sequences that match particular intrusion signature [26]. Anomaly-based or behavior IDS solutions identify particular intrusion signatures to identify and review unusual or malicious behavior patterns[27]. It utilizes machine learning, statistical and artificial intelligence approaches to evaluate huge data amounts and network traffics and detect deviations. Specification-oriented IDS detections also monitor for any anomalies to detect the occurrences of intrusion trends [16][28]. It monitors for their behavior deviations from the normal specifications.

2.7. Android Malware Detection Techniques

As indicated in Fig. 3, hybrid detection, static detection, and dynamic detection are the major Android malware detection classification algorithms.

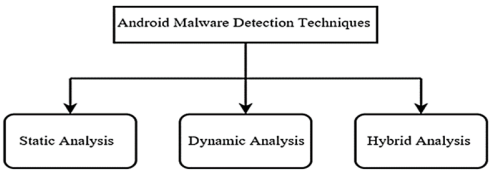

Fig. 3. Android Malware Detection Techniques

The static detection technique handles the attributes that are mined from the appellations or suspect files without execution [29]. This technique analyses

malware files without running the applications, detecting the malware without activating the inbuilt defense mechanisms[7]. Static detection is the safest and most secure approach to examining malware because executing the codes could potentially infect the android systems [18], [29]. Static detections, in their most fundamental forms, glean information from malware without even inspecting the codes. Basic static detections involve analyzing the executable files without observing or programming the real instructions. Fundamental static analyses can verify whether files are malicious, offer information about their functionalities, and sometimes provide data that will enable the android systems to generate modest network signatures [30]. Static analysis can be meaningful in identifying packed files, malicious infrastructures, and libraries[18]. Technical indicators are detected, such as file header data, domains, and strings like IP addresses, hashes, and file names can be utilized to establish whether those files are malicious.

The dynamic detection approaches to monitor and reviews the executed codes' interactions with the systems [16]. Dynamic analyses are examinations conducted after executing malware. Dynamic detection approaches are the second stage in the malware assessment process. The closed system is the process of analyzing and testing programs while the software is running. Also known as dynamic code scanning, dynamic analyses improve the diagnoses and corrections of application crashes, bugs, and memory issues during their implementation [31]. Dynamic malware analyses execute suspected malicious codes in safe environments known as sandboxes. The dynamic analysis allows security experts to look out for the malware in action without the risks of allowing them to escape into the enterprise networks or infect their systems [32]. The advantage of this approach to malware analysis is that it identifies dynamic code loading and records app behavior during the operation period (29). The dynamic analysis takes time, but it is effective against the obfuscation of malware.

The hybrid analysis provides advanced security tools to identify and classify android malware [18]. Hybrid Intrusion Detection (HID) systems offer intrusion detection capabilities by integrating artificial neural networks with advanced pattern recognition engines for effective identification of suspicious activities within a network. Hybrid detection reliably identifies known and unknown vectors and intruders[30]. The integrated system provides a framework for effective filtering and grouping of malware threats while providing routine breach notifications to the users. The security warnings minimize the number of signals sent to the network administrators [33]. The hybrid intrusion detection systems are acquired by integrating Network Traffic Anomaly Detections (NETADs) and Packet Header Anomaly Detections (PHADs). These are anomaly-oriented IDSs with misuse-oriented IDS Snorts that are open-source projects [29]. Investigators prefer combining hybrid detection to enhance malware discoveries because of its dynamic and static detection approaches[34]. The goal of this study is to develop a graph-oriented strategy model for the identification and classification of android malware, applying ML algorithms with low resource requirements.

3. METHODOLOGY

3.1. Methods

A three-pronged research methodology is applied in the detection model. Creating connected flow graphs (CFG) is the first step. CFGs are graphs of the Android malware datasets that identify the classifications. They extract characteristics of classifications after that and select two features of categories, including API calls and permissions.

The second stage is to choose the most critical characteristics by utilizing IPA calls and permissions of graph forms and developing training datasets. The third stage is to produce classifiers based on four particular machine learning algorithms from random forest regressions to K-Nearest neighbor regressions[35], decision tree regressions, and logistic regressions, and then identify malware utilizing the classifiers.

3.2. Schemes

Generally, as explained in Fig. 4, the classical learning strategies have five phases data collection, data pre-processing, feature extractions[36], feature selections, and classifications.

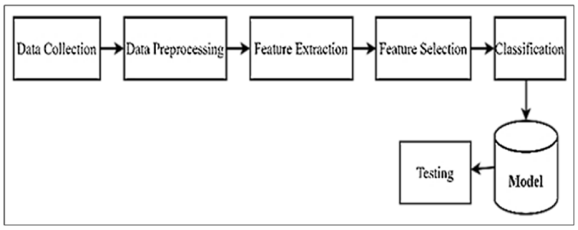

Fig. 4. An Overview of Basic Classical Machine Learning Systems

Phases shown in Fig. 5 are used to construct graph module approaches for Android malware detections. The Fig. 6 illustrates a series of steps followed. Dataset is the first phase. Datasets from the Derbin projects were utilized for this phase. The second

phase is the pre-processing of Android apps. Static characteristics such as system API calls and permissions are obtained in the third phase. The feature selection phase is the fourth phase. The features obtained in the feature mining phase are minimized to small sets of appropriate features in the fourth phase. The fifth phase is the classification phase. Machine learning models are trained in this stage. Then, the machine learning techniques are analyzed utilizing confusion matrix approaches in the final phase. Fig. 6 depicts the entire architecture of this model.

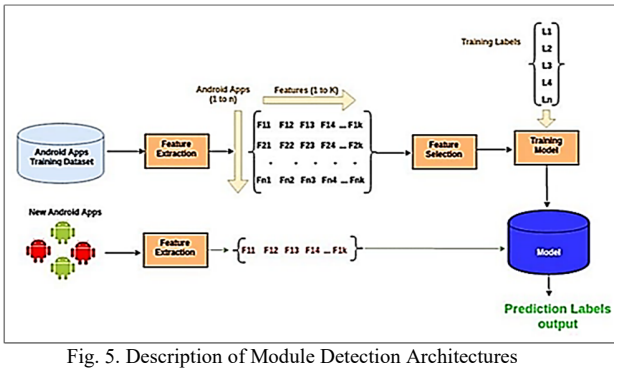

Fig. 5. Description of Module Detection Architectures

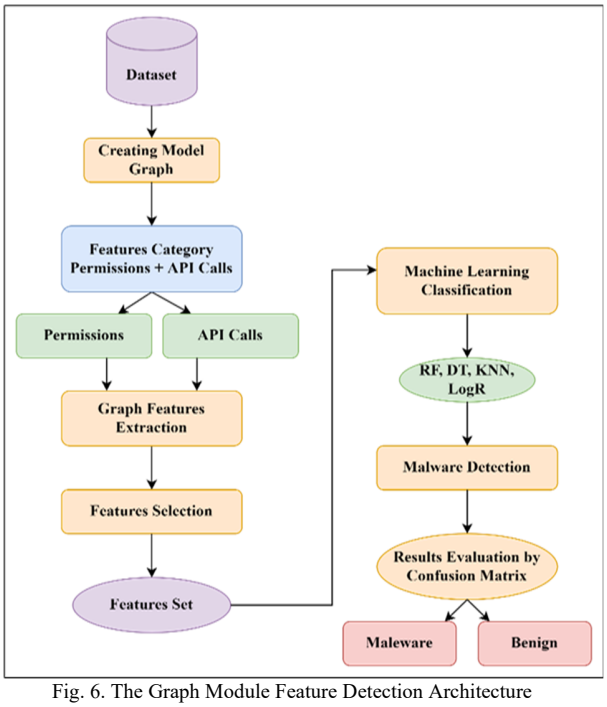

Fig. 6. The Graph Module Feature Detection Architecture

The study utilized a three-pronged methodology in the Android malware detection model. The first step is to create connected flow graphs (CFG) of the Android malware datasets to establish the classifications. The feature categories are then categorized from graphs, and two feature categories are selected with API calls and permissions.

Select is the second phase. The most critical features are extracted by utilizing API calls and graph form permissions. The features are used to establish training datasets. Generating classifiers on the basis of particular four machine learning algorithms is the third stage.

The experiments' goals are to examine the performances of classifiers utilizing characteristics from Android APK files like permission and API calls to design accurate detection techniques of malware [9]. F1-scores, logistic regression, k-nearest neighbor, decision tree, and random forest are the four distinctive machine learning classifiers employed in this study. Better outcomes are obtained by taking more measurements. There are tradeoffs between precisions and recalls, but recalls are more critical than precisions.

The researchers conducted different experiments to analyze the four classifiers and respond to the study questions. The experiments utilized 66% cross-validation and ten-fold split analysis approaches [37]. The study included features selection evaluation experiments and machine learning classifiers evaluation experiments.

3.3. Algorithms

3.3.1. Graph Construction Algorithm

Input:
- Datasets (D), the features set (F) are represented by columns, and the values of Android apps are represented by rows.
- Categories datasets (D2), the rows describe each special feature (Fi) classification (Ci).

Process:
- Extract the columns (F1, F2… Fn) of D in F.
- Let C = (C1, C2… Cn) set of categories in (D2).
- For each category Ci ∈ D2, add node Vi to the set of nodes V if Vi ∈/ V.
- For each feature Fi ∈ D2, add edge Ei to set of edges E if Ei ∈/ E.

Outputs:
- Return graphs G (V, E) for all classification (Ci) permission, intents, command signatures, API calls and other attributes.
- The classes of attributes that compose datasets were identified after applying algorithm one. API calls and data permission are the two sets we concentrate on since they are robust attributes for identifying malware in the Android system.

Second stage:
The attribute selection stage was executed by choosing the most common API calls and permissions in malicious apps and benign to accomplish this.

3.3.2. Graph Extraction Features Algorithm

Input:
- Set of apps (Applications) in datasets (D2)
- Datasets (D2) that have Fi categories ci (APIs and permissions).

Process:
- Starting V ← 0 and E ← 0
- Open the file X // X contain all connected graph node
- For App $F_i \in C_i$ in D2 do // Extract the (F1, F2… Fn)
- Let F = (F1, F2… Fn) set of categories in (D) draw node V
- If w (F_i,…., Fn = = 0)
- Then drop the V (single feature)
- else
- for every Features Fi ∈ X, create edge between every Features node
- if (Fi connected to Fj, Fn) ∈/ E then
- weight W (Fi, Fj…., Fn) = 1 (connected graph edge)
- else
- weight W (Fi, Fj…., Fn) =+1 mean the same connected graph in other Apps
- end if
- end if
- write X in the new dataset D2
- open file Y // contain highest F- Score Form (permissions &API)
- for (Fi ∈ X) do in D2
- use ANOVA filter (Analysis of Variance) SelectKBest method
- Return frequency Graph G (V, E) highest F-score for all features
- write result in file Y
- end for
- end for

Outputs:
- Return frequency graphs G (V, E) greatest F-scores Form (APIs and permissions) in file Y.

3.3.3. Classification Features Algorithm

Input:
Set all Applications (App1, App2, Appn) to be

Process:
- for each App in D do
- Let F = (F1, F2, F3...Fn) be set of Features in X
- if (F=0) or (F=1) then (means=0=benign,1= Malware)
- Return the graph G (V, E). (connected graph)
- else
- Mal =0; Ben=0
- for Every feature in D (Fi, Fj ……, Fn) ∈ X do
- Mal = W (Fi, Fj...Fn) ∈ G(Malware)
- Ben = W (Fi, Fj…., Fn) ∈ G(Benign)
- end for
- if Malware score > Benign score then
- Return App $_i$ as malware
- else
- Return App_i as Benign
- end if
- end if
- end for

Outputs:
- Output every app as normal or malware

4. RESULTS AND DISCUSSION

4.1. Features selection evaluation experiments

The datasets went through various steps, such as cleaning and transforming data. Then, the processed data was utilized to mine extra representative attributes using the graph construction algorithms [38]. The processed data, as a result, was utilized to extract four kinds of attributes: intents, command signatures, API calls, manifest permissions, and other command signs of other features, as indicated in Fig. 7;

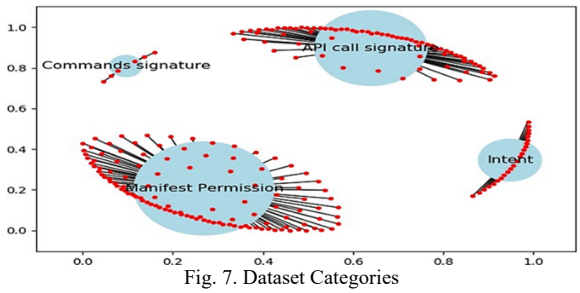

Fig. 7. Dataset Categories

The experiments aimed to examine the characteristics and establish the best vector feature using the graph construction algorithms. Permissions and API calls were the two categories extracted from the datasets (27). The features were selected from the API and permission graphs for each app according to algorithms. The number of every attribute is indicated in Table 1;

TABLE 1. ATTRIBUTES CLASSIFICATIONS

Number	Attributes Classification	Number of features
1	Permissions only	110
2	API call only	73
3	API calls and permissions	183

The ANOVA SelectKBest filter method was used to extract the most important N-features from each application. Fig. 8 and Fig. 9 show that the best features were selected in Table 2 and Table 3.

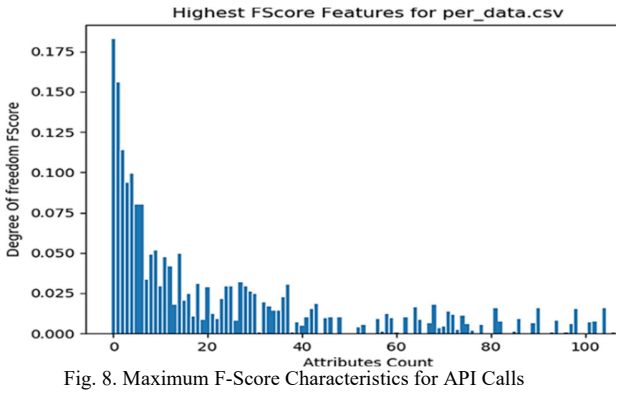

Fig. 8. Maximum F-Score Characteristics for API Calls

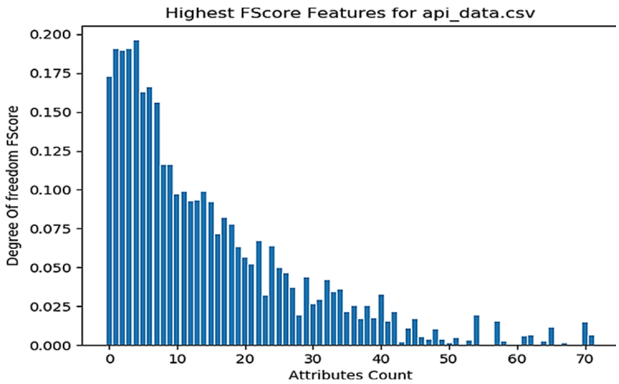

Fig. 9. Maximum F-Score Characteristics for Permissions

TABLE 2. ATTRIBUTE NAMES FOR API CALL AND FREQUENCIES

Attribute Name for API Call	Frequencies
['Ljava.lang.Class.cast']	0.12441
['Ljava.lang.Class.getMethods']	0.16038
['Ljava.lang.Class.getCanonicalName']	0.16718
['Android.os.Binder']	0.17238
['ServiceConnection']	0.17477
['attachInterface']	0.19082
['bindService']	0.19324
['onServiceConnected']	0.19362

TABLE 3. ATTRIBUTE NAMES FOR PERMISSIONS AND FREQUENCIES

Feature Name for Permission	Frequencies
['WRITE_SMS']	0.04694
['MANAGE_ACCOUNTS']	0.07984
['USE_CREDENTIALS']	0.08330
['READ_SMS']	0.08864
['RECEIVE_SMS']	0.09923
['GET_ACCOUNTS']	0.10792
['READ_PHONE_STATE']	0.14790
['SEND_SMS']	0.18394

Besides, attribute classification selections are utilized in evaluating attribute classifications like API calls, permission, and other attributes. Evaluations of attribute classifications were separately performed by testing or training the classifiers on every attribute classification (23). The recall values were calculated and utilized as metrics using a 66% split evaluation technique, as indicated in the Table 4;

TABLE 4. ATTRIBUTE CLASSIFICATIONS AND EXPERIMENTAL RESULTS

Number	Attribute Classifications	Number of Attributes	Recalls
1	Permissions only	110	95
2	API calls only	73	96
3	Permissions and API calls	183	97.3

Android permission is developed to safeguard Android by stopping Android apps from using the devices' hardware without permission from users of the devices or accessing sensitive information and data. The dataset analyses show that benign apps need less permission than malware apps. While about 66.4% of malware apps need over ten permissions, over 90% of benign apps need lesser than ten permissions, as indicated in Fig. 10;

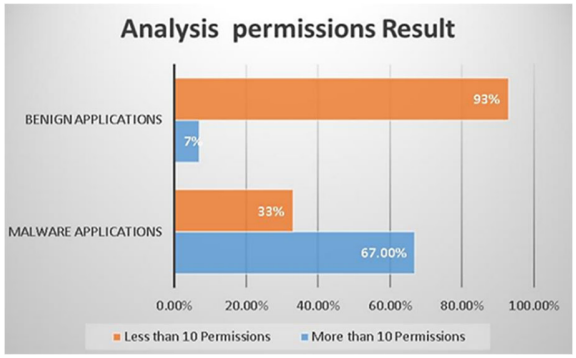

Fig. 10. Analysis Results of Benign App Permissions

4.2. Machine learning classifiers and experiments evaluations

The major study goal is to recommend approaches for detecting malware under Android devices. To accomplish this, the researchers have conducted various steps from datasets to training, testing, and evaluation of machine models. The feature matrices were developed ruing the attribute selection phase. Then, the matrices were utilized for machine learning systems' training, testing, and evaluation. To analyze them, four diverse classifiers were used. Table 5 below shows the four metrics FPR and TPR, computed for this model of supplying the datasets.

TABLE 5. FPR AND TPR OF EXISTING PROPOSED TECHNIQUES

ML Methods	FPR	TPR
Logistic Regressions	60	96
K-Nearest Neighbors	46	97
Random forests	16	96
Decision trees	47	97

The results in this study indicate that the classifiers with the best accuracies were random forests, then k-nearest neighbors, followed by a decision tree, and then logistic regressions while utilizing two diverse splitting datasets.

4.3. Splitting Dataset

Dividing the datasets into percentages was the first technique. This implied that categorization findings were analyzed on the original data subsets [39]. The datasets were then divided for evaluations by 66%. The F-measures and Pre., Recalls, the two metrics, are displayed in Fig. 11;

Fig. 11. F-measures and Accuracies of Splitting Datasets for all Classifiers

While logistic regressions rated the lowest accuracy (95%) in malware detection, the random forest had the greatest accuracy (97%) in malware detection, as shown in Table 6.

TABLE 6. EVALUATION OF ALGORITHMS FOR SPLITTING DATASET VALIDATIONS

ML Algorithms	Precision	Recalls
Logistic Regressions	96	96
K-Nearest Neighbors	97	97
Random forests	99	96.80
Decision trees	97	97

4.4. Cross Validation

The datasets are divided into ten sections known as folds by WEKA [18]. Each part is held in turn. WEKA averages the results to conduct ten-fold cross-validation. The results acquired by using four diverse machine learning category approaches are displayed in the tables and figures below. Fig. 12 and Table 7 show the accuracy percentages accomplished by all four classifiers and indicate that logistic regressions have the least accuracy (86.5%) in malware detection, and the random forest has the greatest accuracy in malware detection (97%). The decision tree and k-nearest reported a 96.50% accuracy level each. Table 8 illustrates that. And Fig. 13 explains the error rates.

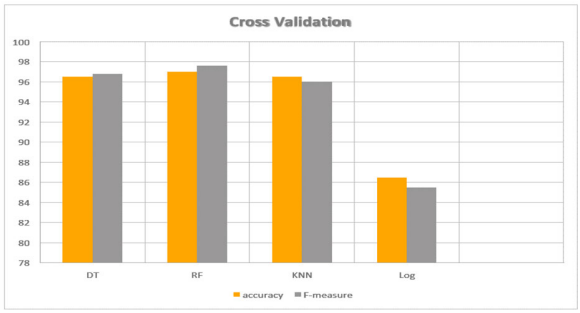

Fig. 12. F-measures and Accuracy of Cross Validations for All Classifiers

TABLE 7. TEN-FOLD VALIDATION OF ALGORITHM EVALUATION

ML Algorithms	Precision	Recalls
Logistic Regressions	83	85.2
K-Nearest Neighbors	94.9	95
Random forests	98	96.2
Decision trees	97	96

TABLE 8. EVALUATION RESULTS OF ALGORITHMS

ML Algorithm	Splitting Datasets		Cross Validations	
	ACCURACIES	*F-MEASURES*	*ACCURACIES*	*F-MEASURES*

Logistic Regressions	95.4	96.5	86.5	85.5
K-Nearest Neighbors	96.6	97.4	96.5	96
Random forests	97.3	97.9	97	97.6
Decision trees	96.5	97.3	96.5	96.8

Fig. 13. The Error Rates

4.5. Processing Time

Processing time is described as the time amount needed to finish the activity that is expressed in seconds. According to the analysis results, the recommended random forest needs less processing time compared to other techniques. Logistic regressions and decision trees need more processing time[35]. It takes fewer periods to categorize malware since the suggested method minimizes calculation complexity to five seconds. Fig. 14 below shows the recommended and existing methods of processing times based on the attribute choice.

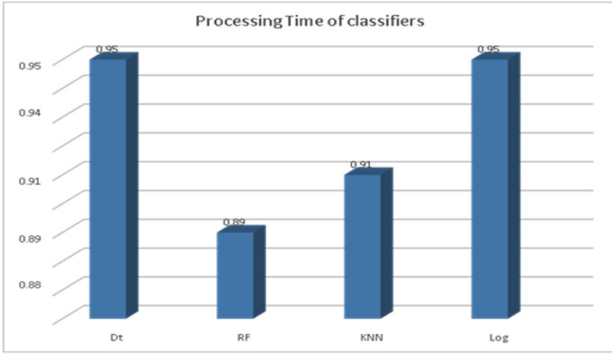

Fig. 14. Classifier Processing Time

5. CONCLUSION

The Android smartphone is greatly vulnerable to malware attacks and intrusions. The Android smartphones are prone to these attacks because of their intrinsic fragility that allows apps to access internal resources when users unintentionally or intentionally get permission. Therefore, the investigators have concentrated on detecting the malicious permission that results in malware identification. Common to malware and normal apps, most permissions present themselves in diverse patterns and lead to intrusions. It is thus important to get a great combination of the permissions that can be hazardous. Static analyses are recommended to identify Android malware in this study. Static techniques focus on feature selection and production utilizing graphs. API calls and permissions were the two natural characteristics to develop new attributes and train the classifiers utilizing machine learning methods. Logistic regressions, decision tree regressions, K-nearest neighbor regressions, and random forest regressions are the four machine learning algorithms utilized to categorize the datasets when they are benign or malicious. The analysis findings show that random forest regressions are the best classification methods for feature sets. The recommended technique also needed less than ten seconds for analyses averagely, accomplishing recall and accuracy of 96.80% and 97.30%, respectively.

Future studies on the implementation of graph-based machine learning algorithms to detect and classify android malware would focus on assessing the performance of integrated dynamic feature extraction techniques on larger datasets to generate privacy protection insights for the growing android users. Since the present study has not assessed variations in the attributes of malware detected using the binary classification approach, future research is recommended to differentiate the impacts of different types of malware on application performance to enable users to adopt issue-specific remediation strategies.

• REFERENCES

[1] P. Yan and Z. Yan, "A survey on dynamic mobile malware detection," *Software Quality Journal*, vol. 26, no. 3, pp. 891–919, 2018, doi: 10.1007/s11219-017-9368-4.

[2] L. He, X. Wang, H. Chen, and G. Xu, "Online Spam Review Detection: A Survey of Literature," *Human-Centric Intelligent Systems*, Jun. 2022, doi: 10.1007/s44230-022-00001-3.

[3] M. S. Adrees, A. E. Karrar, and W. I. Osman, *Adoption of Smart Cities Models in Developing Countries: Focusing in Strategy and Design in Sudan*, vol. 72. 2021. doi: 10.1007/978-3-030-70713-2_84.

[4] M. Alazab, M. Alazab, A. Shalaginov, A. Mesleh, and A. Awajan, "Intelligent mobile malware detection using permission requests and API calls," *Future Generation Computer Systems*, vol. 107, pp. 509–521, 2020, doi: https://doi.org/10.1016/j.future.2020.02.002.

[5] M. Algarni, M. Alkhelaiwi, and A. Karrar, "Internet of Things Security: A Review of Enabled Application Challenges and Solutions," *International Journal of Advanced Computer Science and Applications*, vol. 12, no. 3, 2021, doi: 10.14569/IJACSA.2021.0120325.

[6] A. G. Akintola *et al.*, "Performance Analysis of Machine Learning Methods with Class Imbalance Problem in Android Malware Detection," *International Journal of Interactive Mobile Technologies*, vol. 16, no. 10, pp. 140–162, 2022, doi: 10.3991/ijim.v16i10.29687.

[7] J. Sahs and L. Khan, "A Machine Learning Approach to Android Malware Detection," in *2012 European Intelligence and Security*

Informatics Conference, 2012, pp. 141–147. doi: 10.1109/EISIC.2012.34.

[8] A. G. Akintola *et al.*, "Performance Analysis of Machine Learning Methods with Class Imbalance Problem in Android Malware Detection," *International Journal of Interactive Mobile Technologies*, vol. 16, no. 10, pp. 140–162, 2022, doi: 10.3991/ijim.v16i10.29687.

[9] Z. Ma, H. Ge, Y. Liu, M. Zhao, and J. Ma, "A Combination Method for Android Malware Detection Based on Control Flow Graphs and Machine Learning Algorithms," *IEEE Access*, vol. 7, pp. 21235–21245, 2019, doi: 10.1109/ACCESS.2019.2896003.

[10] F. Martinelli, F. Marulli, and F. Mercaldo, "Evaluating Convolutional Neural Network for Effective Mobile Malware Detection," *Procedia Comput Sci*, vol. 112, pp. 2372–2381, 2017, doi: https://doi.org/10.1016/j.procs.2017.08.216.

[11] A. E. Karrar and M. F. I. Fadl, "Security protocol for data transmission in cloud computing," *International Journal of Advanced Trends in Computer Science and Engineering*, vol. 7, no. 1, 2018, doi: 10.30534/IJATCSE/2018/01712018.

[12] M. Spreitzenbarth, F. Freiling, F. Echtler, T. Schreck, and J. Hoffmann, "Mobile-Sandbox: Having a Deeper Look into Android Applications," in *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, 2013, pp. 1808–1815. doi: 10.1145/2480362.2480701.

[13] "Stephen A. Ridley," in *Tribe of Hackers*, John Wiley & Sons, Ltd, 2019, pp. 208–212. doi: https://doi.org/10.1002/9781119643395.ch53.

[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, 2019, doi: 10.1109/ACCESS.2019.2919796.

[15] I. U. Haq, T. A. Khan, A. Akhunzada, and X. Liu, "MalDroid: Secure DL-enabled intelligent malware detection framework," *IET Communications*, Jun. 2021, doi: 10.1049/cmu2.12265.

[16] S. and S. S. and C. R. Raghuraman Chandni and Suresh, "Static and Dynamic Malware Analysis Using Machine Learning," in *First International Conference on Sustainable Technologies for Computational Intelligence*, 2020, pp. 793–806.

[17] H. Yuan, Y. Tang, W. Sun, and L. Liu, "A detection method for android application security based on TF-IDF and machine learning," *PLoS One*, vol. 15, no. 9 September, Sep. 2020, doi: 10.1371/journal.pone.0238694.

[18] J. T. McDonald, N. Herron, W. B. Glisson, and R. K. Benton, "Machine learning-based android malware detection using manifest permissions," in *Proceedings of the Annual Hawaii International Conference on System Sciences*, 2021, vol. 2020-January, pp. 6976–6985. doi: 10.24251/hicss.2021.839.

[19] X. Jiang and Y. Zhou, *Android Malware*. 2013. doi: 10.1007/978-1-4614-7394-7.

[20] M. Kakavand, M. Dabbagh, and A. Dehghantanha, "Application of machine learning algorithms for android malware detection," Nov. 2018. doi: 10.1145/3293475.3293489.

[21] Y. Liu, K. Guo, X. Huang, Z. Zhou, and Y. Zhang, "Detecting Android Malwares with High-Efficient Hybrid Analyzing Methods," *Mobile Information Systems*, vol. 2018, 2018, doi: 10.1155/2018/1649703.

[22] A. P. Felt, M. Finifter, E. Chin, S. Hanna, and D. Wagner, "A survey of mobile malware in the wild," in *Proceedings of the ACM Conference on Computer and Communications Security*, 2011, pp. 3–14. doi: 10.1145/2046614.2046618.

[23] G. D'Angelo, F. Palmieri, and A. Robustelli, "A federated approach to Android malware classification through Perm-Maps," *Cluster Comput*, Aug. 2022, doi: 10.1007/s10586-021-03490-2.

[24] A. Karrar and K. Dahbur, *Computing Ethics*. New York: Nova Science Publishers, Inc., 2021. [Online]. Available: http://www.scopus.com/inward/record.url?eid=2-s2.0-85109665455&partnerID=MN8TOARS

[25] S. Wang *et al.*, "Krrecover: An auto-recovery tool for hijacked devices and encrypted files by ransomwares on android," *Symmetry (Basel)*, vol. 13, no. 5, May 2021, doi: 10.3390/sym13050861.

[26] O. A. Alzubi, J. A. Alzubi, A. M. Al-Zoubi, M. A. Hassonah, and U. Kose, "An efficient malware detection approach with feature weighting based on Harris Hawks optimization," *Cluster Comput*, vol. 25, no. 4, pp. 2369–2387, 2022, doi: 10.1007/s10586-021-03459-1.

[27] M. Elmubarak, A. Karrar, and N. Hassan, "Survey in Anomaly and Misuse Intrusion Detection System," *IOSR Journal of Engineering*, vol. 9, p. 65, Jun. 2019.

[28] Mohamed Elmubarak, Abdelrahman Karrar, and Nafeesa Hassan, "Implementation Hybrid (NIDS) System using Anomaly Holtwinter Algorithm and Signature based Scheme," *International Journal of Advances in Scientific Research and Engineering (IJASRE), ISSN:2454-8006, DOI: 10.31695/IJASRE*, vol. 5, no. 6, pp. 141–148, Jun. 2019, doi: 10.31695/IJASRE.2019.33278.

[29] M. Dhalaria and E. Gandotra, "Android Malware Detection Techniques: A Literature Review," *Recent Patents on Engineering*, vol. 14, Jul. 2020, doi: 10.2174/1872212114999200710143847.

[30] M. Ijaz, M. H. Durad, and M. Ismail, "Static and Dynamic Malware Analysis Using Machine Learning," in *2019 16th International Bhurban Conference on Applied Sciences and Technology (IBCAST)*, 2019, pp. 687–691. doi: 10.1109/IBCAST.2019.8667136.

[31] J. Tang and H. Zhao, "AmandaSystem: A new framework for static and dynamic Android malware analysis," *Journal of Intelligent & Fuzzy Systems*, vol. Preprint, pp. 1–15, 2022, doi: 10.3233/JIFS-220567.

[32] T. S. John and T. Thomas, "Evading Static and Dynamic Android Malware Detection Mechanisms," in *Communications in Computer and Information Science*, 2021, vol. 1364, pp. 33–48. doi: 10.1007/978-981-16-0422-5_3.

[33] A. Rodríguez-Mota, P. J. Escamilla-Ambrosio, and M. Salinas-Rosales, "Malware Analysis and Detection on Android: The Big Challenge," in *Smartphones from an Applied Research Perspective*, InTech, 2017. doi: 10.5772/intechopen.69695.

[34] A. Martín, R. Lara-Cabrera, and D. Camacho, "Android malware detection through hybrid features fusion and ensemble classifiers: The AndroPyTool framework and the OmniDroid dataset," *Information Fusion*, vol. 52, pp. 128–142, 2019, doi: https://doi.org/10.1016/j.inffus.2018.12.006.

[35] A. E. Karrar, "The Effect of Using Data Pre-Processing by Imputations in Handling Missing Values," *Indonesian Journal of Electrical Engineering and Informatics (IJEEI)*, vol. 10, no. 2, Apr. 2022, doi: 10.52549/ijeei.v10i2.3730.

[36] M. Umair *et al.*, "Main path analysis to filter unbiased literature," *Intelligent Automation and Soft Computing*, vol. 32, no. 2, 2022, doi: 10.32604/iasc.2022.018952.

[37] M. A. Jerlin and K. Marimuthu, "A New Malware Detection System Using Machine Learning Techniques for API Call Sequences," *Journal of Applied Security Research*, vol. 13, no. 1, pp. 45–62, 2018, doi: 10.1080/19361610.2018.1387734.

[38] A. E. Karrar, "Investigate the Ensemble Model by Intelligence Analysis to Improve the Accuracy of the Classification Data in the Diagnostic and Treatment Interventions for Prostate Cancer," *International Journal of Advanced Computer Science and Applications*, vol. 13, no. 1, 2022, doi: 10.14569/IJACSA.2022.0130122.

[39] A. E. Karrar, "A Proposed Model for Improving the Performance of Knowledge Bases in Real-World Applications by Extracting Semantic Information," *International Journal of Advanced Computer Science and Applications*, vol. 13, no. 2, 2022, doi: 10.14569/IJACSA.2022.0130214.