Pathak *et al.*
EURASIP Journal on Information Security (2024) 2024:33
https://doi.org/10.1186/s13635-024-00182-3

EURASIP Journal on Information Security

RESEARCH

Open Access

Static analysis framework for permission-based dataset generation and android malware detection using machine learning

Amarjyoti Pathak[1*], Th. Shanta Kumar[2] and Utpal Barman[3]

Abstract

Since Android is the popular mobile operating system worldwide, malicious attackers seek out Android smartphones as targets. The Android malware can be identified through a number of established detection techniques. However, the issues presented by modern malware cannot be met by traditional signature or heuristic-based malware detection methods. Previous research suggests that machine-learning classifiers can be utilised to analyse permissions, making it possible to differentiate between malicious and benign applications on the Android platform. There exist machine-learning methods that utilise permission-based attributes to build models for the detection of malware on Android devices. Nevertheless, the performance of these detection methods is dependent on the raw or feature datasets. Android malware research frequently faces a major obstacle due to the lack of adequate and up-to-date raw malware datasets. In this paper, we put forward a systematic approach to generate an Android permission-based dataset using static analysis. To create the dataset, we collect recent raw malware samples (APK files) and focus on the reverse engineering approach and permission-based features extraction. We also conduct a thorough feature analysis to determine the important Android permissions and present a machine-learning-based Android malware detection mechanism. The experimental result of our study demonstrates that with just 48 features, the random forest classifier-based Android malware detection model obtains the best accuracy of 97.5%.

Keywords Android malware detection, Static analysis, Permission feature extraction, Feature engineering, Machine learning

1 Introduction

According to Statista web portal [1], in the global mobile operating system market, Android continues to hold the top spot with a market share of 70.7% in the first quarter of 2024. Several elements are involved in this dominance, including the following: (i) since its open source, installing and customising it cost nothing [2], (ii) the capacity to add plenty of programs from the official application market (the Play Store) to increase the operating system's default functionality, and (iii) in addition to Google, the Android OS is developed and distributed by a group of over 84 software development companies (OHA — Open Handset Alliance), such as Sony, Samsung, and HTC [3]. Unfortunately, because of its enormous popularity, malware program developers are attracted to it. Fraudulent APK (Android Application Package) files can take control

*Correspondence:
Amarjyoti Pathak
amar.pathak@gmail.com
[1] GIMT, Guwahati under Assam Science and Technology University, Guwahati, Assam, India
[2] Department of CSE, Girijananda Chowdhury University, Guwahati, Assam, India
[3] Faculty of Computer Technology, Assam down town University, Guwahati, Assam, India

of Android smartphones through malware and cause significant harm to them [4].

Antivirus software providers and the antimalware research community offer the initial layer of defence against any malware attack. In the beginning, signature-based detection mechanisms served as the foundation for antivirus engines. A signature is a particular pattern that is kept up to date in an antivirus database and serves to identify malware uniquely. But the main problems with signature-based methods are that they are not scalable, and they are vulnerable to 0-day attacks [5]. Heuristic engines are frequently used in antivirus programs to supplement signature-based detection, in which malware specialists create rules to identify potentially harmful activity on a system [6]. However, it is difficult to write specific criteria for such engines to detect malicious activity without raising the false-positive rate [7].

In recent years, scholars have started to look at developing malware detection mechanisms based on machine learning [8]. There are two stages involved in building these systems: feature extraction using application analysis and classification. Based on the traditional research methodology described in the literature, experts and researchers commonly utilise three analysis techniques—static, dynamic, and hybrid—for feature extraction to detect Android malware [9]. Static analysis gathers fundamental data about the functionality of the application to investigate malware without running the actual code. Dynamic analysis, on the other hand, tracks an application's nature and looks for indications of malicious behaviour. The hybrid study combines static and dynamic analysis. Even though code obfuscation and other complex malicious transformation techniques provide some challenges for static analysis, it is still a very effective, practical, and widespread method of detecting malware before it executes. Static features in Android are taken from the APK file's source code. Of all the static features, permissions is the most widely applied and popular one [10]. Several publications addressing the use of static malware analysis have been released in earlier research. For instance, Urcuqui Lopez et al. [11] suggested a framework for static analysis and used machine learning to distinguish between legitimate and dangerous apps. They gathered 558 APKs in total and evaluated them using a list of 330 attributes. Using K-neighbours, SVM, and the decision tree algorithm, they reached the best accuracy of 94% in classification. DREBIN [12] conducts a thorough static analysis, extracting as many features (API, permissions, network address, etc.) as it can from the manifest and program of an application. With a detection rate of 93.9%, DREBIN offers a decent performance.

Google incorporated a permission system into the Android OS, which requires all developers to specify the permissions needed for their application's functioning. However, the choice to allow or prohibit access to the requested permission rests with the user. Thus, it appears that monitoring and analysing permissions can help prevent Android malware from spreading. In recent trends, researchers have developed reliable machine-learning models for detecting Android malware by analysing the significance of permissions [13–17]. Sanz et al. [17] extracted the application permissions and utilised machine-learning algorithms to detect malware. A total of 1811 benign and 249 malicious programs were used to evaluate their framework, called PUMA. With an accuracy of 86.41%, the random forest (RF) classifier produced the best results. A. P. Felt et al. [14] introduced Stowaway, a tool that identifies over-privilege in Android programs that have been developed. Stowaway consists of two components: an API call that lists the permissions required for each API call and a static analysis tool that establishes the permission map of an application. Santosh K. et al. [15] devised a system that identifies the most significant features using the feature reduction technique. They used a dataset containing 398 samples and 330 permissions — 198 of which are malware and 198 of which are benign. Gain ratio, information gain, and ReliefF were used to compare various feature reduction techniques. The authors achieved the maximum accuracy of 93.46% using the randomizable filtered classification method utilising the gain ratio when the top 5 permissions were considered for performance evaluation. D. O. Sahin et al. [16] proposed a novel machine-learning-based malware detection approach for Android to differentiate between goodware and malware applications. The authors employed a feature selection technique using linear regression to eliminate superfluous permissions. They experimented with 1000 malicious and 1000 legitimate apps. They downloaded malware from the Android malware dataset as well as useful apps from APKPure. The authors used the SMO algorithm to obtain the best performance of 0.9655 (F-measure) (102 permissions) without carrying out the feature selection procedure. Using the same approach, they achieved the greatest accuracy of 0.961 (F-measure) (27 permissions) by implementing feature selection.

Based on the review of the literature, we observe that most works used datasets from 2009 to 2020. These datasets, which include samples from earlier Android system versions, were also used by several recent articles published after 2020, such as Şahin et al. [16], Sihag et al. [18], and Sarah et al. [19]. As new malware emerges in its behaviour patterns, it is important to update the malware features in the dataset. To ensure reliable evaluation of malware detection systems, it is required to periodically develop unbiased malware datasets. From the literature

survey, we observe that many researchers' uses older datasets in their research work. Older datasets are never enough to assess how well the defence mechanism is working. In this study, we attempt to suggest a mechanism to detect Android malware that is both reliable and efficient. We gather the latest malware samples directly from the sources that are regularly updated with the newest malware. We use the static analysis technique to construct a framework to generate a novel permission-based dataset. The comprehensive permissions list is manually compiled from the official Android docs.

We further carry out a feature vector analysis utilising the feature reduction technique to minimise number of features in the dataset. Feature reduction is the act of lowering a dataset's feature count, or dimensions, while preserving as much information as feasible. It helps machine learning algorithms computationally efficient and reduces storage.

Eventually, we assess the performance with prominent machine-learning classifiers, such as K-NN, Naive Bayes, decision tree, and random forest.

The following highlights the key contributions of this work:

1) We suggest a framework for generating a featured dataset through static analysis. This method gathers malware samples from a live malware repository and uses reverse engineering to extract permission features from the gathered APK files.
2) We propose an Android malware detection system that uses machine learning-based classification models after thorough feature engineering (using a feature importance-based attribute reduction strategy).
3) The random forest method works better and reaches a 96.25% accuracy rate with full permission. Our reduced feature models, built with just 15% of the Android permissions, attain maximum accuracy while saving a significant amount of time. Despite having fewer features, the random forest classifier achieves an accuracy of 97.5%.

2 The proposed system

This study aims to propose an effective and efficient Android malware detection system. Figure 1 depicts the proposed systematic structure of the empirical study used to classify malware. The architecture is broken down into the following modules:

2.1 Permission-based dataset generation

The dataset preparation framework outlines the entire process of getting ready for the created dataset. We separate them into three phases: data collection, data analysis and feature extraction, and feature vector generation. In the data collection phase, Android applications that are malicious and benign are gathered for additional examination. The data gathered in phase 1 is subjected to static data analysis employing a reverse engineering approach and feature extraction in the second phase. Lastly, phase 3 creates a binary variable for each extracted permission and gives each application a label to produce the final dataset.

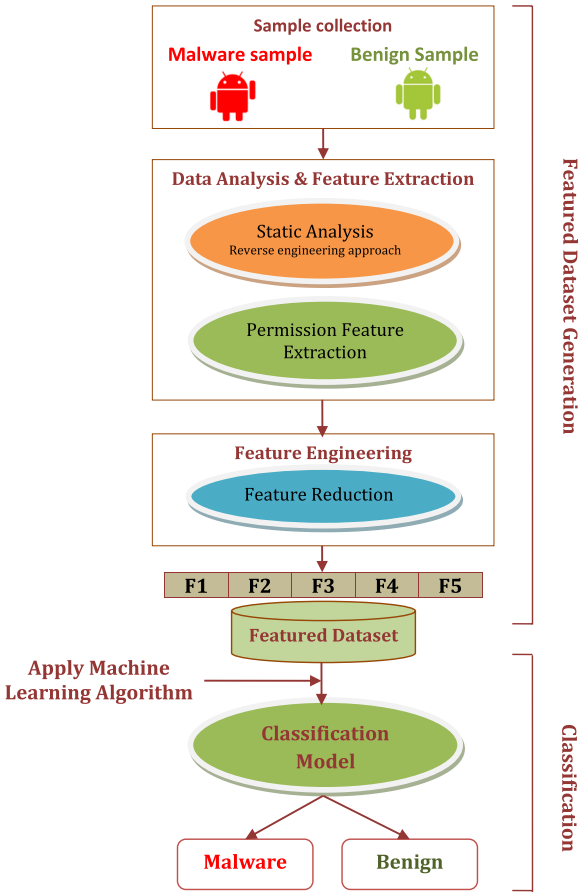

Fig. 1 Proposed Android malware detection system

For our experimental study, we collect over 700 Android applications from various sources. We extract the Android permissions from the application that could serve as features for our models. We carefully compile 323 permissions from the official Android developer website. The end product of this module is a permission-based dataset.

2.2 Classification models

The primary goal of machine learning-based classification is to create a model that can correctly classify new Android applications. Various machine-learning algorithms can be employed to develop the detection models. We assess the Android malware detection system's

performance using four common machine learning algorithms, viz. K-nearest neighbour (K-NN), Naïve Bayes (NB), decision tree (DT), and random forest (RF). Later on, each module is covered in detail.

3 Methodology

This section covers the steps involved in creating the permission-based dataset, feature reduction methods applied to the dataset, and several machine learning algorithms utilised in the development of an effective Android malware detection system.

3.1 Permission-based dataset

Any program that wants to access the restricted data, whether it be malware or goodware, must initially go via the permission authorisation process. We predominantly preferred permission-based static features in our study since permissions are typically the first thing malicious software will attempt to exploit.

3.1.1 Data collection

To construct a featured dataset, we need application package kits (APKs) including both benign and malicious applications. We use two sources for malware sample collection — MalwareBazaar database [20] and VirusShare [21]. MalwareBazaar is a project from abuse.ch where the latest malware samples are added every day. VirusShare is a malware sample repository that offers access to live malware samples for security researchers, incident responders, and forensic analysts. To determine whether the APK files we download are malicious or not, we use VirusTotal [22]. We gather benign samples from the Google Play Store [23], which is regarded as the official marketplace for Android applications. We collected a total of 585 Android APK files, out of which 385 are malware and 200 are benign.

3.1.2 Data analysis and permission feature extraction

The analysis of data takes place in an isolated environment. VirtualBox is employed to create an isolated environment, with Kali Linux installed as the guest operating system for all analysis tasks. In this phase, we use a reverse engineering approach for data analysis. We utilise Apktool, which is included in Kali Linux, to decompile APK files and make the AndroidManifest.xml file accessible. All application permissions needed to run the programme are contained in this AndroidManifest.xml file. We develop a Python program to parse the XML file, extract all permissions from the Manifest file, and store them in a.csv file. The pseudo-code snippet is shown in Fig. 2.

3.1.3 Feature vector

After collecting the data, we randomly selected 398 Android applications (199 malicious and 199 benign) to create the feature vector and permission-based dataset. Using the official Android developer website [24], we compile a comprehensive list of 323 permissions. Depreciated permissions are included in this list. This stage involves handling the analyser to produce a binary variable. The presence or absence of a permission in the analysed AndroidManifest.xml file determines the value associated with that permission in the list. Let V be a vector that represents every selected permission feature. For each Android app that is downloaded, we generate a binary sequence $V_i = \{F1, F2, F3, ..., F_j\}$ and as follows:

$$F_j = \begin{cases} 1\ \textit{The analyser detected the permission} \\ 0\ \textit{Otherwise} \end{cases} \quad (1)$$

We write a Python program to generate the feature vector; a pseudo-code is displayed in Fig. 3. A label pertaining to the application's category is generated concurrently with the permissions analysis.

```
Input: AndroidManifest.xml
Output: Permission_features_list
1. procedure permission_extraction(AndroidManifest.xml)
2. root ← parse("AndroidManifest.xml")
3. permissions ← root.findall("uses-permission")
4. permission_feature_list = [ ]
5. for perm in permissions do
6.         for attribute in perm.attrib do
7.                 permission_feature_list.append(perm.attrib[attribute])
8. return permission_feature_list
9. end procedure
```

Fig. 2 Pseudo-code for performing the permission extraction

```
Input: Permission_features of an application, List of all permissions
Output: Feature vector of one application
1. procedure feature_vector(Permission_features of an application, List of
   all permissions)
2. Store the Permission features list into a DataFrame named DFPermission
3. Store the list of all permissions into a DataFrame named DFvector.
4. append_value = 0, append_list = [ ]
5. for column in DFvector do
6.         append_value = 0
7.         for index, row in DFPermission.itertuples() do
8.             if column == row do
9.                 append_value = 1
10.        append_list.append(append_val)
11. Store append_list values in DFvector
12. return DFvector
13. end procedure
```

Fig. 3 Pseudo-code for feature vector generation

$$l_i = \begin{cases} 1 \; if \; the \; application \; is \; malware \\ 0 \; if \; the \; application \; is \; benign \end{cases} \quad (2)$$

Finally, we combine the feature vector and the associated label for all applications to form the permission-based dataset, which we store as a CSV file.

3.2 Feature engineering

A preliminary examination of the feature vector provided some insightful information about the utilisation of Android permissions. We observe that there is variation in the distribution of permission features. Ten significant permissions and their usage frequency in the dataset are displayed in Fig. 4. Applications that contain malware frequently use certain permissions, while those that are benign use others. Malicious and legitimate apps use Android permissions like ACCESS_NETWORK_STATE and INTERNET somewhere equally. However, malware applications are more inclined to use permissions like READ_SMS, READ_PHONE_STATE, and READ_CONTACTS.

Moreover, of the 323 permissions in the dataset, neither malware applications nor benign applications have used 146 of them. For malware and benign applications separately, the figure is 169 and 197, respectively.

A correlation is a connection or relationship whereby two variables are likely to change whenever one of them does. Figure 5 displays the relationship between

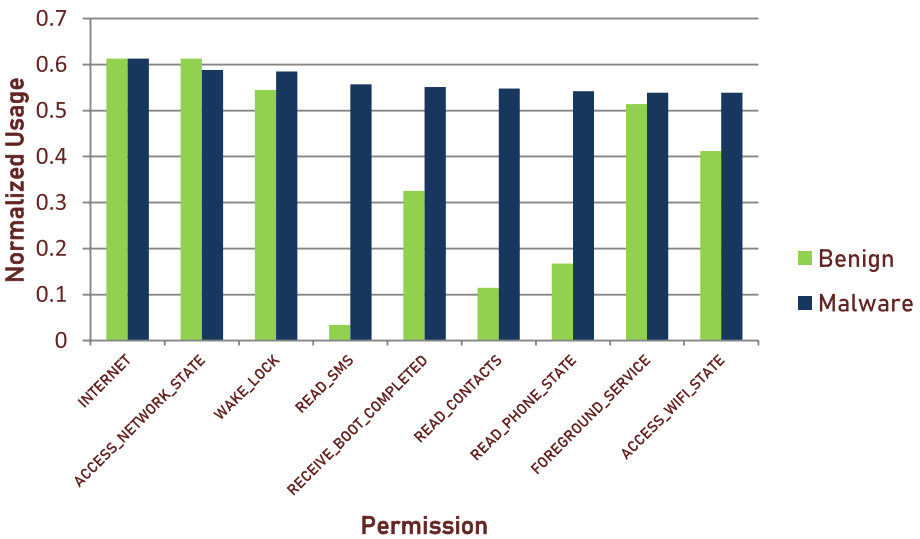

Fig. 4 Android permission usage frequency in malicious and benign applications

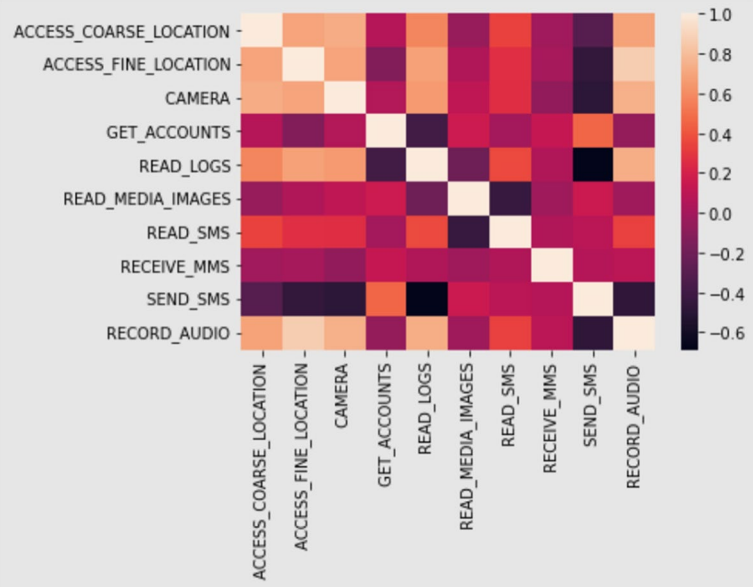

Fig. 5 Permission correlation matrix utilised in malicious software

the top 10 dangerous permissions in malicious applications that are obtained by gradient boosting. The greatest correlation is 0.86 between RECORD_AUDIO and ACCESS_FINE_LOCATION, followed by 0.74, 0.73, and 0.73 between CAMERA and RECORD_AUDIO, CAMERA and ACCESS_COARSE_LOCATION, and CAMERA and READ_LOGS. Malware frequently uses READ_SMS and READ_CALL_LOG to obtain crucial personal data. With the first permission, the app can access details about sent and received SMS messages; with the second permission, it can access the device's call history. These permissions can be combined in many ways to accomplish the harmful task that is intended.

3.2.1 Feature reduction

Our feature vector makes use of 323 Android permissions. Large feature counts in a classification model will make the system more computationally expensive. Moreover, training the model takes longer on big datasets with a huge feature set. Thus, we use the feature subset selection approach (feature importance score) to achieve feature reduction [25].

Feature importance score is a feature reduction strategy that involves calculating each feature's relevance score inside a dataset and removing features with lower scores from the final vector. Our computation of the feature importance score for Android permissions reveals that 274 out of 323 features have a feature importance value of zero. With a feature importance value of 0.283, permission READ_SMS has the highest rating. The feature importance score of the top 50 Android permissions is highlighted in Fig. 6. Based on our analysis, we remove all Android permissions from the feature list if they have a feature relevance score of 0. With just 48 features, we can finally create our reduced feature vector. The top 20 Android permissions are listed in Table 1 along with a feature importance score.

3.3 Malware classification models

To create an Android malware detection system, this study uses four standard classification algorithms. These algorithms are K-nearest neighbour (K-NN), Naïve-Bayes (NB), decision tree (DT), and random forest (RF).

3.3.1 K-nearest neighbour (K-NN) algorithm

A popular supervised machine-learning technique for classification is K-nearest neighbour. It calculates the separation between classes and a data point. The adjacency with the points in the training set determines the value prediction of a new data point. To calculate distance across all datasets, this study employs the default metric. By default, "Minkowski" is used. When the power parameter, p, equals 2, the conventional Euclidean distance is obtained. $K=5$ is selected as the nearest neighbour number (K) since it yields the best accuracy with these values.

3.3.2 Naïve-Bayes algorithm

The algorithm is supervised and is founded on the Bayes theorem. The Bayes theorem provides a method for calculating the posterior probability. The Naïve-Bayes

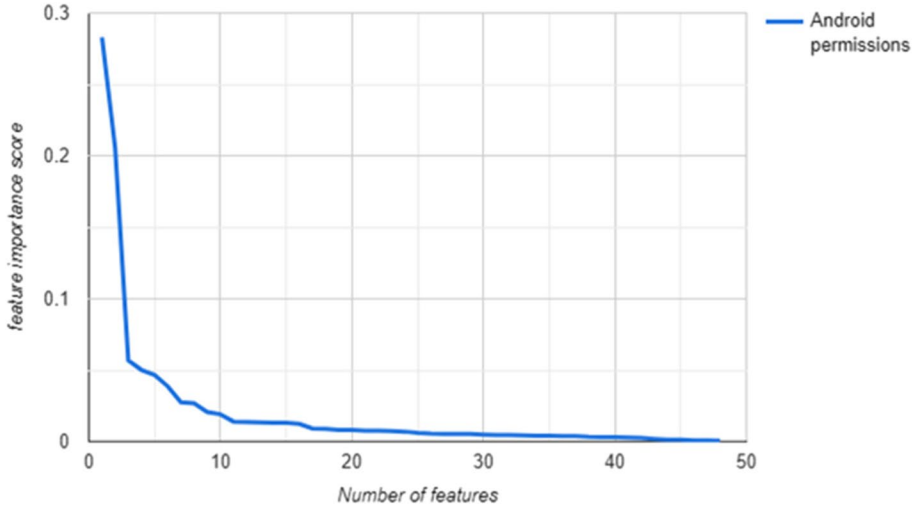

Fig. 6 Feature importance score of top 48 Android permissions

Table 1 Top 20 important permissions

Permissions	Importance score
READ_SMS	0.282916278
FOREGROUND_SERVICE_LOCATION	0.205475003
USE_FINGERPRINT	0.056946058
READ_SYNC_SETTINGS	0.050180726
RECEIVE_MMS	0.046693586
READ_LOGS	0.038653832
SCHEDULE_EXACT_ALARM	0.027465085
INSTALL_SHORTCUT	0.026794842
SEND_SMS	0.020696118
BLUETOOTH_CONNECT	0.019080613
GET_TASKS	0.013990888
MODIFY_AUDIO_SETTINGS	0.013627682
GET_ACCOUNTS	0.013404317
MOUNT_UNMOUNT_FILESYSTEMS	0.013162673
BLUETOOTH_ADMIN	0.013102572
BLUETOOTH	0.012360611
CAMERA	0.009172391
ACCESS_COARSE_LOCATION	0.008857181
MANAGE_OWN_CALLS	0.008126509
ACCESS_FINE_LOCATION	0.008015067

classifier assumes that one feature's existence does not imply another's presence. To apply the algorithm to the dataset, this work uses default settings.

3.3.3 Decision tree algorithm

A supervised learning method called decision-tree is generally employed to address categorisation issues. The decision node and leaf node make up a decision tree. While leaf nodes are the result of decisions made by decision nodes and do not include any additional extensions, decision nodes make decisions with multiple branches. To get the best outcome in this study, the criterion = "entropy" is applied.

3.3.4 Random forest algorithm

Among the supervised learning, algorithms mainly utilised for classification is the random forest algorithm. Using data samples, it builds decision trees, extracts predictions from each tree, and uses voting to select the best outcome. This model is an ensemble one. This study measures the split quality using estimators = 100 and criterion = "entropy". Applying the given criterion yields the maximum accuracy.

3.4 Performance metrics

An essential part of any machine-learning process is assessing the model's output. In this procedure, the trained model forecasts previously unseen tagged data. Classification evaluates the percentage of these predictions that the model accurately predicted. Most of the time, models cannot be completely accurate in real-world classification scenarios. Therefore, knowing how and in what way a model was incorrect is helpful when assessing it.

A classification model is assessed using the following four values: true positive (TP), false positive (FP), true negative (TN), and false negative (FN). Numerous measures that assess performance can be obtained from these anticipated values. This study measures the performance of the classifier using precision, recall, accuracy, and F1 score.

The easiest way to calculate accuracy is to take the ratio of accurate forecasts to total predictions.

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN} \quad (3)$$

The precision is the proportion of true positives that the classifier has actually detected out of all samples that have been classified as positive.

$$Precision = \frac{TP}{TP + FP} \quad (4)$$

The recall is the ratio of samples that the classifier correctly classified as positive overall to samples that were actually classified as negative.

$$Recall = \frac{TP}{TP + FN} \quad (5)$$

F1 score is the harmonic mean of precision and recall values.

$$F1score = \frac{2 \times \text{precision} \times \text{recall}}{\text{Precision} + \text{recall}} \quad (6)$$

4 Results and discussion

The primary scripting language utilised in this study is Python 3.12. The libraries and functions offered by Python are extensive. Throughout several stages of analysis, operating systems differ. Our preinstalled operating system is Windows 10. We use the virtualization technique with Oracle VirtualBox to establish an isolated environment for the purpose of collecting malware samples. The guest operating system for all analysis operations is Kali Linux, which is configured inside VirtualBox. An Intel Core i5-8265U CPU with 16 GB RAM and a 256 GB SSD is used in the experiment.

We will first discuss the performance of the selected classification models. The performance with the entire feature set (323 features) is displayed in Table 2. The random forest classification algorithm, with an accuracy score of 96.25%, achieves the best accuracy when all features are considered.

Next, to minimise the feature set size, we employ the feature reduction technique utilising the feature importance score. The performance of the classifiers with a reduced feature set (48 features) is displayed in Table 3. The random forest algorithm obtains the best accuracy of 97.5% with the reduced feature set.

Additionally, we conducted a comparison of the classification models' accuracy prior to and following feature reduction. Table 4 displays the outcomes of the comparison. Accuracy losses with the K-NN classification model, gains in accuracy with the Naive-Bayes and random forest algorithms, and neither gain nor loss with the decision tree method are noted. Based on observations, the random forest classification model yields the maximum accuracy.

Our customised feature reduction technique reduces the feature set size by about 85%. Figure 7 displays the size of the feature set before and after feature reduction. The classification algorithms' execution times are displayed in Fig. 8. We see a significant improvement in the algorithm's overall running time. A comparison of our approach with some of the previous research that uses attribute selection in Android malware detection is given in Table 5. The data shows that the random forest classifier yields the best results in most studies, and

Table 2 Performance metric with full feature (323 features)

ML models and measures	Accuracy score	Precision	Recall	F1 score
KNN classification model	90.00%	97.05%	82.50%	89.18%
Naïve-Bayes classification model	62.50%	14.70%	83.30%	25.00%
Decision tree classification model	95.00%	94.11%	94.11%	94.11%
Random forest classification	96.25%	100.00%	91.89%	95.77%

Table 3 Performance metric with reduced feature (48 features)

ML models and measures	Accuracy score	Precision	Recall	F1 score
KNN classification model	86.25%	97.05%	76.74%	85.71%
Naïve-Bayes classification model	86.25%	94.11%	78.04%	85.33%
Decision tree classification model	95.00%	94.11%	94.11%	94.11%
Random forest classification	97.50%	100.00%	94.40%	97.14%

Table 4 Comparison (accuracy gain or loss)

ML models and measures	With the full feature set	With the reduced feature set	Accuracy gain/loss
KNN classification model	90.00%	86.25%	−3.75%
Naïve-Bayes classification model	62.50%	86.25%	23.75%
Decision tree classification model	95.00%	95.00%	0.00%
Random forest classification	96.25%	97.50%	1.25%

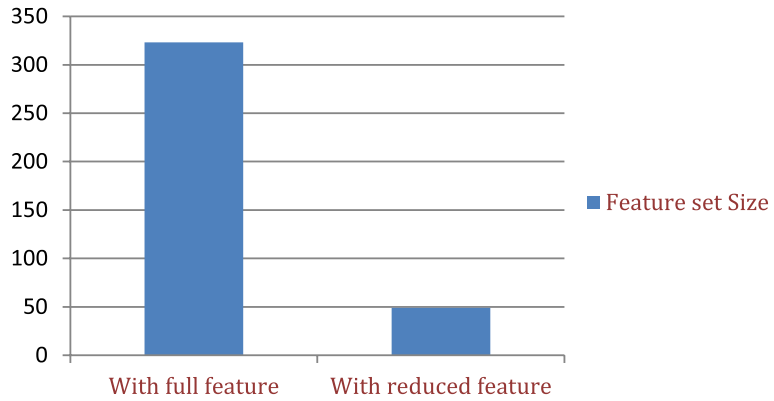

Fig. 7 Comparison of feature set size

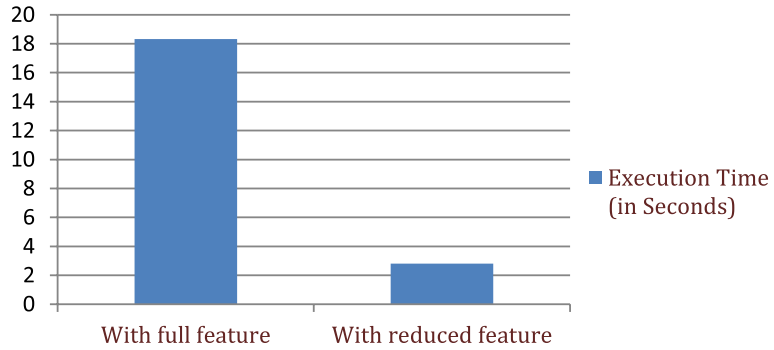

Fig. 8 Model execution time

our method uses only 48 features to obtain the greatest accuracy of 97.5%.

Dangerous permissions for Android are requests made by an application during runtime that provide access to view private information or carry out banned actions. Although the runtime permission model strengthens the security of the Android system, many Android apps in reality have a variety of runtime problems [30, 31]. Nevertheless, these problems with runtime permissions can only arise when the application is running. All developers using Android OS are required to specify the list of permissions required for their applications to function or to correctly invoke the Android API. As a result, the list of all Android permissions needed to operate the programme effectively is contained in the AndroidManifest.xml file. Our suggested method for creating datasets is essentially permission-based static analysis, which looks into malware without running the programme. It uses the AndroidManifest.xml file to get the fundamental data about the operation of the app. A reverse engineering approach is used to study the Android application's source code and extract its features. As a result, the suggested method is unaffected by Android runtime permission concerns because all attributes are extracted without running the code.

Table 5 Comparison of performance with earlier work

Study	Dataset size	ML classifier	Feature selection method	Performance
Sanz et al [17]	1811 benign 249 malware	Random forest	Permission tag	86.41% (accuracy)
Sahin et al [26]	199 malware 200 benign	K-nearest neighbour	Relevance frequency	96.63% (accuracy)
Santosh K. et al (K. et al., 2020)	199 malware 199 benign	Randomizable filtered classification	Gain ratio	93.46% (accuracy)
A. Sangal et al [27]	1126 benign 396 malware	Random forest	PCA	96.05% (accuracy)
Rathore et al [28]	5560 malware 5721 benign	Random forest	Variance threshold	93.3% (accuracy)
A. Shatnawi et al. [29]	1126 benign 396 malware	Support vector machine	RFE	94.36% (accuracy)
D. O. Sahin et al [16]	1000 malware 1000 benign	Random forest	Linear regression	96.1% (F-measure)
Our approach	199 malware 199 benign	Random forest	Feature importance score	97.5% (accuracy)

The issue with permission-based datasets is that when new viruses and Android versions emerge, the datasets are out of current, and the detection system can no longer function. One way to solve this is to retrain the classifier using an updated dataset. When to retrain a malware detection system is a challenging decision, though [32]. Dangerous permissions pose the greatest threat to Android security, given the level of protection afforded by Android permission systems. According to official releases from the Android platform, harmful permissions do not change over the course of several Android version releases, even though Android permissions are constantly evolving. Consequently, our model will remain viable until a new Android update adds any dangerous permissions.

Dangerous permissions are crucial in permission-based malware detection systems. We can make our system as resilient as other cutting-edge malware detection systems by researching the pattern of development of dangerous permissions in Android. We can include this in our study's future scope.

Because it is easy to extract static features, static approaches are typically simpler to implement. Nevertheless, there are several drawbacks to static methods, and as a result, models constructed using Android permissions are typically found to be unable to identify the hidden behaviour of dynamic code loading and code obfuscation. Consequently, some malicious apps might use subtle techniques to avoid being detected based on permissions. Using permissions as a feature is also limited by the fact that many detection models cannot track behaviours that do not result in permission checks.

5 Conclusion and future works

The number of Android systems has increased dramatically during the past 10 years. Because Android devices are so popular, cybercriminals target them. According to research, heuristic- and signature-based detection engines are unable to handle new-generation malware. Machine learning-based malware detection has become the most popular defence strategy against Android malware in recent years. As new malware appears and changes its behaviour patterns, it is imperative to update the malware features in the dataset. It is never feasible to assess the defensive mechanism's efficacy using older datasets. This research focuses on the process of creating permission-based datasets using recent malware samples. The entire process of generating the permission-based dataset is thoroughly and methodically explained, beginning with data collecting, reverse engineering, permissions extraction from the AndroidManifest.xml file, and feature vector construction. Our analysis shows that feature engineering results in an effective detection system for Android malware that improves accuracy and shortens the model execution time. Our experimental result reveals that with feature reduction, just 48 permissions are needed to create a classification model that gives us 97.5% accuracy, saving a substantial amount of time throughout the model training and test stages. Future research in this field will investigate how useful various feature selection techniques are. Furthermore, besides traditional machine learning algorithms, we will also apply deep learning techniques to improve classification performance.

Acknowledgements
We express our gratitude to the Department of Computer Science and Engineering, Girijananda Chowdhury University, for providing the necessary facilities and support.

Authors' contributions
The authors confirm their contribution to the paper as follows: Amarjyoti Pathak: Literature survey, study conception and design, malware sample collection, static analysis and featured dataset generation, machine learning classification algorithm and feature analysis, analysis and interpretation of results, and draft manuscript preparation. Th. Shanta Kumar: Literature survey, study conception and design, analysis and interpretation of results, and supervision. Utpal Barman: Machine learning classification algorithm and feature analysis, analysis and interpretation of results, and supervision. All authors reviewed the results and approved the final version of the manuscript.

Funding
Not applicable.

Data availability
The dataset generated and/or analysed during the current study is available from the corresponding author upon reasonable request.

Declarations

Ethical approval and consent to participate
Not applicable.

Consent for publication
Not applicable.

Competing interests
The authors declare no competing interests.

Received: 2 September 2024 Accepted: 14 October 2024
Published online: 23 October 2024

References

1. A. Sherif, Mobile OS market share worldwide 2009–2024. Statista. (2024). Available at: https://www.statista.com/statistics/272698/global-market-share-held-by-mobile-operating-systems-since-2009/. Accessed 24 July 2024
2. M. Yang, S. Wang, Z. Ling, Y. Liu, Z. Ni, Detection of malicious behavior in android apps through API calls and permission uses analysis. Concurr. Comput. **29**, e4172 (2017). https://doi.org/10.1002/cpe.4172
3. A.T. Kabakus, What static analysis can utmost offer for Android malware detection. ITC **48**, 235–240 (2019). https://doi.org/10.5755/j01.itc.48.2.21457
4. K. Liu, S. Xu, G. Xu, M. Zhang, D. Sun, H. Liu, A review of Android malware detection approaches based on machine learning. IEEE Access **8**, 124579–124607 (2020). https://doi.org/10.1109/ACCESS.2020.3006143
5. Ye Y, Li T, Adjeroh D, Iyengar SS (2017) A survey on malware detection using data mining techniques. ACM. Comput. Surv. 50:41:1–41:40. https://doi.org/10.1145/3073559
6. Z. Bazrafshan et al., A survey on Heuristic Malware Detection Techniques', in *The 5th Conference on Information and Knowledge Technology [Preprint]*. (2013). https://doi.org/10.1109/ikt.2013.6620049
7. P. Faruki, A. Bharmal, V. Laxmi, V. Ganmoor, M.S. Gaur, M. Conti, M. Rajarajan, Android security: a survey of issues, malware penetration, and defenses. IEEE Communications Surveys & Tutorials **17**, 998–1022 (2015). https://doi.org/10.1109/COMST.2014.2386139
8. D. Gibert, C. Mateu, J. Planes, The rise of machine learning for detection and classification of malware: research developments, trends and challenges. J. Netw. Comput. Appl. **153**, 102526 (2020). https://doi.org/10.1016/j.jnca.2019.102526
9. J. Senanayake, H. Kalutarage, O. Al-Kadri, Android mobile malware detection using machine learning: a systematic review. Electron. **10**(13), 34 (2021). https://doi.org/10.3390/electronics10131606
10. Y. Sharma, A. Arora, A comprehensive review on permissions-based Android malware detection. Int. J. Inf. Secur. **23**, 1877–1912 (2024). https://doi.org/10.1007/s10207-024-00822-2
11. Urcuqui-López C, Cadavid AN (2016) Framework for malware analysis in Android. Sistemas y Telemática 14:45–56. https://doi.org/10.18046/syt.v14i37.2241
12. D. Arp et al., Drebin: Effective and explainable detection of Android malware in your pocket, in *Proceedings 2014 Network and Distributed System Security Symposium [Preprint]*. (2014). https://doi.org/10.14722/ndss.2014.23247
13. F. Akbar, M. Hussain, R. Mumtaz, Q. Riaz, A.W.A. Wahab, K.-H. Jung, Permissions-based detection of Android malware using machine learning. Symmetry **14**, 718 (2022). https://doi.org/10.3390/sym14040718
14. A.P. Felt et al., Android permissions demystified, in *Proceedings of the 18th ACM conference on Computer and communications security*. (2011). pp 627–638. https://doi.org/10.1145/2046707.2046779
15. K. SJ, S. Chakravarty, P.R.K. Varma, Feature selection and evaluation of permission-based Android Malware Detection, in *2020 4th International Conference on Trends in Electronics and Informatics (ICOEI) (48184)*, vol. 21, (2020), pp.795–799. https://doi.org/10.1109/icoei48184.2020.9142929
16. D.Ö. Şahin, O.E. Kural, S. Akleylek, E. Kılıç, A novel permission-based Android malware detection system using feature selection based on linear regression. Neural Comput. & Applic. **35**, 4903–4918 (2023). https://doi.org/10.1007/s00521-021-05875-1
17. B. Sanz, I. Santos, C. Laorden, X. Ugarte-Pedrero, P.G. Bringas, G. Álvarez, PUMA: permission usage to detect malware in Android, in *International Joint Conference CISIS'12-ICEUTE´12-SOCO´12 Special Sessions*. (Springer, Berlin Heidelberg, Berlin, Heidelberg, 2013), pp.289–298
18. V. Sihag, M. Vardhan, P. Singh, BLADE: robust malware detection against obfuscation in Android. Forensic Science International: Digital Investigation **38**, 301176 (2021). https://doi.org/10.1016/j.fsidi.2021.301176
19. N.A. Sarah, F.Y. Rifat, Md.S. Hossain, H.S. Narman, An efficient Android malware prediction using Ensemble machine learning algorithms. Procedia. Comput. Sci. **191**, 184–191 (2021). https://doi.org/10.1016/j.procs.2021.07.023
20. Malware Sample Exchange (no date) MalwareBazaar. Available at: https://bazaar.abuse.ch/. Accessed 27 July 2024
21. J.M. Roberts, VirusShare.com. (2011). Available at: https://virusshare.com/. Accessed 27 July 2024
22. VirusTotal, Virustotal. (2012). Available at: https://www.virustotal.com/gui/home/upload. Accessed 27 July 2024
23. Android apps on Google Play (no date) Google. Available at: https://play.google.com/store/games?hl=en. Accessed 27 July 2024
24. Android mobile App Developer tools (no date) Android Developers. Available at: https://developer.android.com/. Accessed 28 July 2024
25. Pathak A, Barman U, Kumar ThS (2024) Machine learning approach to detect android malware using feature-selection based on feature importance score. J. Eng. Res. S2307187724000981. https://doi.org/10.1016/j.jer.2024.04.008
26. Sahin DO, Kural OE, Akleylek S, Kilic E (2018) New results on permission based static analysis for Android malware. In: 2018 6th International Symposium on Digital Forensic and Security (ISDFS). IEEE, Antalya, pp 1–4
27. Sangal A, Verma HK (2020) A static feature selection-based Android malware detection using machine learning techniques. In: 2020 International Conference on Smart Electronics and Communication (ICOSEC). IEEE, Trichy, India, pp 48–51
28. H. Rathore et al., Identification of significant permissions for efficient Android malware detection, in *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*. (2021), pp 33–52. https://doi.org/10.1007/978-3-030-68737-3_3
29. A.S. Shatnawi, Q. Yassen, A. Yateem, An Android malware detection approach based on static feature analysis using machine learning algorithms. Procedia. Comput. Sci. **201**, 653–658 (2022). https://doi.org/10.1016/j.procs.2022.03.086
30. M. Dilhara, H. Cai, J. Jenkins, Automated detection and repair of incompatible uses of runtime permissions in Android apps, in *Proceedings*

of the 5th International Conference on Mobile Software Engineering and Systems. (2018), pp 67–71. https://doi.org/10.1145/3197231.3197255
31. Ying Wang et al., Runtime permission issues in Android apps: Taxonomy, practices, and Ways Forward. IEEE Trans. Softw. Eng. **49**(1), 185–210 (2023). https://doi.org/10.1109/tse.2022.3148258
32. K. Xu et al., DroidEvolver: Self-evolving Android Malware Detection System, in *2019 IEEE European Symposium on Security and Privacy (EuroS&P) [Preprint]*. (2019). https://doi.org/10.1109/eurosp.2019.00014

Publisher's Note