Manzil and Manohar Naik *Cybersecurity (2023)6:6*
https://doi.org/10.1186/s42400-023-00139-y

Cybersecurity

RESEARCH

Open Access

Android malware category detection using a novel feature vector-based machine learning model

Hashida Haidros Rahima Manzil* and S. Manohar Naik

Abstract

Malware attacks on the Android platform are rapidly increasing due to the high consumer adoption of Android smartphones. Advanced technologies have motivated cyber-criminals to actively create and disseminate a wide range of malware on Android smartphones. The researchers have conducted numerous studies on the detection of Android malware, but the majority of the works are based on the detection of generic Android malware. The detection based on malware categories will provide more insights about the malicious patterns of the malware. Therefore, this paper presents a detection solution for different Android malware categories, including adware, banking, SMS malware, and riskware. In this paper, a novel Huffman encoding-based feature vector generation technique is proposed. The experiments have proved that this novel approach significantly improves the efficiency of the detection model. This method makes use of system call frequencies as features to extract malware's dynamic behavior patterns. The proposed model was evaluated using machine learning and deep learning methods. The results show that the proposed model with the Random Forest classifier outperforms some existing methodologies with a detection accuracy of 98.70%.

Keywords Android malware, Dynamic analysis, Malware category, Huffman coding

Introduction

The legendary Android operating system has dominated the smartphone industry since 2011 (Statista 2011). The Android operating system has approximately 2.5 billion active users from 190 countries, according to Android Statistics (Business of Apps: Android Statistics 2022). In this digital era, Android smartphones play an essential role in fulfilling multiple user needs. Therefore, its impacts on various aspects of society are immense. The Android operating system dominated the global market share since 2014 and it loomed over 87% of the market share in 2022 (Business of Apps: Android Statistics 2022). Android has become a prime target for cybercriminals because of its widespread use and open-source nature. Cybercriminals and anti-malware developers are constantly at odds in the realm of malware detection. With evolving technologies, the two protagonists have quickly modified their strategies. Malicious actors typically strive to profit in an unethical or even unlawful way. Mobile malware could steal sensitive and confidential user data, misuse the user's device to send SMS to premium text services, or install adware that causes users to view malicious websites or download other malware. The researchers have conducted several studies to develop countermeasures against Android operating systems and application security issues. Generally, most of the studies are based on the detection of generic Android malware, and studies based on the malware categories are relatively few. The malicious patterns can effectively be identified by the malware category recognition. Android malware detection can

*Correspondence:
Hashida Haidros Rahima Manzil
hashidahydros@gmail.com
Department of Computer Science, Central University of Kerala, Kasaragod 671316, Kerala, India

be categorized into the signature and behavior-based detection. The signature-based approach detects malicious behavior only after malware attacks have already occurred (a posteriori event), because it generates well-defined patterns (Portokalidis et al. 2006; Wressnegger et al. 2017; Oyama et al. 2012). Also, signature-based solutions rely on cryptographic algorithms or similarity measurement techniques (Tchakounté et al. 2021). This traditional method successfully detects well-known dangerous patterns because these malicious patterns are already stored in the database, whereas it cannot detect zero-day attacks.

On the other hand, the behavior-based approaches are further classified into static, dynamic, and hybrid methods. Researchers frequently use static analysis to extract features without running applications on a real device or emulator. This approach is particularly appealing since it requires less computing time and overhead during implementation. Static analysis cannot detect malware that hides or obfuscates its abusive behavior during execution. Dynamic analysis is required to address this issue because it monitors the run-time behavior of the applications. System calls are the most frequently retrieved features by dynamic analysis techniques. System call sequences will adequately reveal the malignant behavior patterns of different malware categories. As a result, this paper presents a dynamic analysis-based method that uses system call frequencies as features. This method successfully identified several categories of Android malware, including riskware, banking trojan, SMS malware, and adware.

The highlights of the proposed methodology are described below:

- The system call frequencies are utilized to build a detection solution for Android malware categories.
- A dynamic analysis-based model is proposed for the detection of Android malware categories.
- A novel feature vector generation method based on Huffman Encoding is incorporated with the detection model.
- The proposed model was evaluated using machine learning and deep learning techniques and compared with previous studies.

Sect. "Related works" summarizes the related studies on Android malware detection. Sect. "Proposed methodology" thoroughly explains the proposed methodology for detecting Android malware categories. The experimental results are discussed in Sect. "Experiments and Results". Sect. "Conclusion" concludes the paper and discusses potential directions for further research.

Related works

The detection systems can recognize more malware by identifying their related families, prioritizing the risky families, and capturing their impact on users (Alswaina and Elleithy 2020). Generally, Android malware detection approaches are classified into two categories: signature-based and behavior-based techniques. The behavior-based detection approaches are further divided into static, dynamic, and hybrid analysis-based techniques.

The signature-based techniques primarily rely on the known signature patterns of the malware. For example, a set of semi-supervised algorithms for the automatic generation of different Android malware family signatures were developed by Atzeni et al. (2018). However, this approach fails to detect unknown malware attacks. Therefore, researchers generally prefer something other than this traditional technique like behavior-based strategies, which includes static, dynamic, and hybrid methods.

The articles (Alswaina and Khaled 2018; Arindaam Roy. et al. 2020; Zhou et al. 2020; Zhu et al. 2021; Elayan and Mustafa 2021; Imtiaz 2021; Almahmoud and Dalia Alzu'bi et al. 2021; Pei et al. 2020; Kim et al. 2021; Bai et al. 2021) propose static analysis-based techniques for detecting Android malware. The studies (Alswaina and Khaled. 2018, Imtiaz 2021, Pei et al. 2020, Kim et al. 2021, Bai et al. 2021) discusses the family classification of Android malware with a static analysis approach. Alswaina et al. (Alswaina and Khaled. 2018) used machine learning approaches to categorize malware families and developed a reverse engineering framework to extract permissions. Ibrahim et al. (2021) proposed DeepAMD for android malware and its family in static and dynamic layers. Similarly, for malware detection and family attribution, Pei et al. (2020) developed a unique deep-learning system called AMalNet. Kim et al. (2021) propose leveraging built-in custom permissions and machine learning to categorize Android malware families. Bai et al. (2021) used static features such as permissions, API calls, activity, services, broadcast receivers, and content providers to classify Android malware families. This study (Bai et al. 2021) uses a lot of machine learning and neural network techniques, based on manual features from literature reviews and documentary features from Android developers.

The dynamic analysis methodology has been applied to detect Android malware since obfuscated malware and malicious dynamic content loading cannot be detected by static analysis. Several research articles, such as those in Mahindru and Sangal (2021a, 2021b), Martín et al. (2018), Abderrahmane et al. (2019), D'Angelo et al. (2021), explore dynamic analysis-based Android malware detection. Martin A et al. (2018) introduce CANDYMAN, a malware classification tool that leverages the

Markov chain to categorize Android malware families. They rely on deep learning techniques and use Markov chains for detection. System calls are used as features in convolutional neural networks by Abderrahamane et al. (2019) to detect fraudulent Android applications. This approach depends on pair-level system call dependencies. With the assistance of the CuckooDroid sandbox (2020), a mobile security framework for static and dynamic feature extraction, D'Angelo et al. (2021) created another dynamic analysis-based Android malware classification system.

The combined static and dynamic features are promoted in hybrid-based detection systems. For example, in their article, Ding et al. (2021) present a hybrid analysis-based technique for identifying Android malware and categorizing malware families. This approach used static and dynamic analysis to extract static features (like permissions and intents) and dynamic features (like network traffic data) to classify malware families. Similarly, Taheri et al. (2019) presented a hybrid, two-layer Android malware analyzer based on static and dynamic analysis-based malware category classification. Dhalaria and Gandotra (2021) depicted another hybrid approach for both malware detection and family classification, in which the authors have employed the information gain feature selection algorithm. Many malware detection studies have recently tried to use machine learning to make advancements in detecting unidentified Android malware (Meijin et al. 2022). El Fiky et al. (2021) developed machine learning-based approaches for identifying Android malware categories. Zhang et al. (2019) propose a combination of n-gram analysis and online classifiers for Android malware detection and family attribution. Shao et al. (2021) introduced a novel detection technique based on sampling strategies. The authors created two distinct sampling algorithms based on various malware families, to address the sample imbalance in the dataset. The original authors of CICMalDroid dataset, Samaneh Mahdavifar et al. (2022) employed a semi-supervised learning method combined with a pseudo-labelling technique. It is obvious that pseudo-labelling is sensitive to the initial predictions. Although this approach reduces label dependency, it may lead to incorrect prediction results if there are only limited data points available. Lee introduced pseudo-labelling technique (Lee 2013). In this technique, clustering is done to label unknown data points. If there are only a few labelled points and proper clustering cannot be performed, the resulting pseudo-labels may lead the classifier to the incorrect decision boundary. The authors claim that their proposed method shows an accuracy of 95.19% with only 100 labelled training samples. However, these results could be unstable with only 100 labelled samples since pseudo-labelling is highly influenced by the initial predictions. Moreover, the semi-supervised technique is highly based on a self-training approach. The main drawback of this approach is that wrong predictions with high confidence will propagate the prediction error into model learning.

Proposed methodology

This section outlines the proposed methodology to classify Android malware categories. The proposed method follows a dynamic analysis-based approach that utilizes system call frequencies as features. Figure 1 represents the model of proposed detection solution. The detailed descriptions of each phase are provided in the following Sects. "Data acquisition", "Data pre-processing" and "Feature selection").

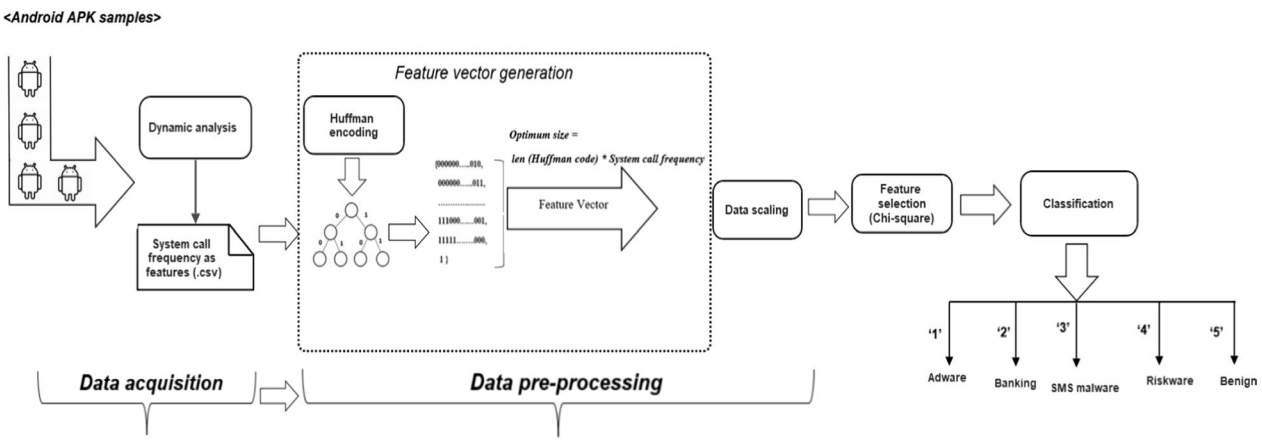

Fig. 1 Proposed system model

Table 1 Count of different application samples

APK sample categories	Count
Adware	1253
Banking	2100
SMS Malware	3904
Riskware	2546
Benign	1795
Total	11,598

Data acquisition

This section summarizes the dataset used for the proposed method. The dataset CICMalDroid (2020), Mahdavifar et al. (2022), Canadian Institute for Cybersecurity (2020) is used, which comprises Android samples broken down into five categories: Adware, Banking malware, SMS malware, Riskware, and Benign. For example, the adware category contains families like Judy, Ewind, Copycat, GhostClicker, etc. Each malware category has a variety of families. This dataset was gathered from different sources, including the VirusTotal service (2022), Contagio Mini Dump blog (2022), MalDozer (Karbab et al. 2018), etc. The proposed methodology employs the CSV (Comma Separated Value) file containing 139 extracted system call frequencies from 11,598 APK (Android Application Package) files of five malware categories. Table 1 shows the count of different APK sample categories used in the study.

Data pre-processing

The act of transforming unprocessed data into something that a machine learning model can use is known as data pre-processing. It is the first and most crucial stage in developing a machine-learning model. The data preprocessing is used to improve the model's accuracy and efficiency. The basic steps in data pre-processing include importing libraries and datasets, finding missing data, removing NULL values or unnecessary data, encoding categorical data, data scaling, augmentation, and feature vector generation. Since there are no missing or NULL values in the dataset, a new Huffman encoding-based feature vector generation technique is proposed and data scaling is applied as pre-processing tasks.

Feature vector generation

The Huffman encoding-based feature vector generation is used in this phase. According to the literature survey, the methodology presented in this paper is the first truly innovative way for detecting Android malware. The system calls help to monitor the dynamic behavior of apps; therefore, it will be easy to identify malicious patterns by determining how frequently a given application uses a system call. System call frequencies are therefore used as features in the study. The proposed method uses the frequencies of 139 different system calls, which were acquired in the data acquisition phase. Then Huffman encoding is used, which is an optimization technique that maps system call frequencies into an optimized size value in O(nlogn) times. The new feature vector is then created using these Huffman-encoded values, which boosts the performance of the detection framework even more due to its higher encoding speed and effectiveness. The Huffman's optimality or minimum-redundancy code property makes it a more efficient technique (Moffat 2019).

Huffman encoding David A. Huffman invented the Huffman Encoding compression technique (Huffman_coding 2022). This technique is based on the frequency of occurrence of a data item. According to this encoding scheme, a unique code is obtained for each system call. Given 'm' number of application samples, $A = \{a_1, a_2, a_3, ..., a_m\}$ and $S = \{s_1, s_2, s_3, ..., s_n\}$ represents the 'n' number of features (system call frequencies) used by 'm' APK samples (Here, $m = 11{,}598$ and $n = 139$) as shown in Table 2

Table 2 Application samples and features

	S_1	S_2	S_3	S_4	…	S_{n-1}	S_n
	_arm_nr_cacheflush	_arm_nr_set_tls	_llseek	_newselect	…	ugetrlimit	umask
A_1	0.0	14.0	6.0	0.0	…	0.0	0.0
A_2	1590.0	42.0	6.0	0.0	…	2.0	0.0
A_3	0.0	23.0	6.0	0.0	…	0.0	0.0
A_4	0.0	27.0	6.0	0.0	…	0.0	0.0
…	……	……	…		…	……	….
A_m	0.0	47.0	204.0		…	1.0	0.0

$$F_{a,s} = \{ f_{a1,s1}, f_{a1,s2}, f_{a1,s3}, \ldots, f_{a1,sn},$$
$$f_{a2,s1}, f_{a2,s2}, f_{a2,s3}, \ldots, f_{a2,sn},$$
$$\ldots\ldots\ldots\ldots\ldots\ldots\ldots,$$
$$f_{am,s1}, f_{am,s2}, f_{am,s3}, \ldots, f_{am,sn} \}$$

Fig. 2 Feature set of proposed system

Then the feature set, $F_{a,s}$ represents all system call frequencies of all APK samples (Fig. 2).

The corresponding Huffman trees for each row of system calls are built and mapped to unique codes by assigning 0 and 1 in the left and right child trees, respectively. At the end, a sequence of 0's and 1's will be generated for each leaf node. Table 3 represents the mapped Huffman codes of some of the given system calls.

As the final step, the optimum size required for each system call frequency value is obtained by multiplying the length of the corresponding Huffman codes with the value of system call frequency (Eq. 1). This will be used as the new feature vector for the final detection model. The pseudo-code of Huffman encoding is shown in Fig. 3.

$$Size = len\big(Huffmancode\big(Systemcall\big)\big) * System\_call\_frequency \tag{1}$$

a. Minimum-redundancy code / Optimality property Minimum-redundancy code or optimality means the average number of coding digits per feature is minimized. This property makes Huffman an efficient technique. Any application of Huffman's algorithm will always create a minimum-redundancy code (Huffman 1952). Therefore, it will help to generate an optimal feature vector for the final detection model. Thus, this can improve the detection accuracy of the model. A minimum-redundancy code exists in which the two least-value features are siblings and share a common parent in the corresponding binary code tree. These two features are joined into a combined node with weight given by their sum. Following that, a minimum-redundancy code is created for this reduced-by-one feature set. Then expanding that feature into its two components, yields a minimum-redundancy code for the original set of features (Moffat 2019).

Data scaling

One of the most critical phases in data pre-processing before building a machine learning model is feature scaling or data scaling. It is used to generalize data points so there will be less space between them. A machine

Table 3 Huffman codes of system call frequencies

System call	Huffman code
Wait4	00,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,111,100
Vfork	00,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,111,101
Access	0,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,001,111,110
Accept	0,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,001,111,111
Unlink	000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,001
Setsid	00,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,001
Setrlimit	0,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,001
setresuid32	00,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,001
Setitimer	0,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,001
Chdir	000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,001

```
def Huffman_Feature_Vector (System_call):
        read {system_call_freq_a , system_call_freq_b}(apk_i)
        if (system_call_freq_a < system_call_freq_b):
                left_node = system_call_freq_a
                right_node = system_call_freq_b
        else:
                left_node = system_call_freq_b
                right_node = system_call_freq_a
        root_node = call makeTree (left_node, right_node)
        huffman_code = call huffmanCodeTree (root_node)
        size = len(huffman_code) * System_call
        return size
def makeTree (left_node, right_node):
        internal_node = left_node + right_node
        left_child = left_node
        right_child = { right_node , internal_node }
        return internal_node
def huffmanCodeTree (internal_node, binString = ' '):
        if internal_node == left_child:
                binString = '0'
        else:
                binString = '1'
        huffman_code = {internal_node, binString}
        return huffman_code
```

Fig. 3 Pseudo code of Huffman encoding

learning model's strength can be changed through scaling, from poor to better. In this work, standard and Min–max scaling techniques are employed.

Feature selection

Feature selection is necessary for a model to predict the target variable. This process aims to minimize the number of input variables to select those features that are identified as most beneficial. The proposed model uses the Chi-square technique to select appropriate features. Thus, it reduces the feature space for the final machine learning model.

Classification

In this phase, the Android malware category classification is experimented using machine learning and deep learning techniques. The machine learning classifiers use the feature vector as input that was obtained from the previous step. Then classifiers like Random Forest, Decision Tree, Logistic Regression, Support Vector Machine, and AdaBoost are employed to detect Android malware

Table 4 Results without Proposed Feature vector generation

Model used	A	P	R	F1-Score
RF	**0.931**	**0.931**	**0.931**	**0.931**
DT	0.897	0.897	0.897	0.897
LR	0.756	0.756	0.756	0.756
SVM	0.804	0.804	0.804	0.804
NB	0.504	0.505	0.505	0.505
KNN	0.875	0.875	0.875	0.875
AdaBoost	0.786	0.786	0.786	0.786
MLP	0.890	0.890	0.890	0.890
CNN	0.7545	0.754	0.754	0.754

A-Accuracy, P-Precision, R-Recall, RF-Random Forest, DT-Decision Tree, LR-Logistic Regression, SVM-Support Vector Machine, NB-Naïve-Bayes, KNN-K Nearest Neighbour, MLP-Multi-Layer Perceptron, CNN-Convolutional Neural Networks

The highest performance in terms of Accuracy, Precision, Recall, and F1-score are highlighted in bold

Table 5 Results with Huffman encoding

Model used	A	P	R	F1-Score
RF	**0.9870**	**0.987**	**0.987**	**0.987**
DT	0.9797	0.980	0.980	0.980
LR	0.7614	0.761	0.761	0.761
SVM	0.8103	0.810	0.810	0.810
NB	0.5247	0.525	0.525	0.525
KNN	0.9038	0.904	0.904	0.904
AdaBoost	0.7717	0.772	0.772	0.772
MLP	0.8911	0.891	0.891	0.891
CNN	0.8129	0.813	0.813	0.813

A-Accuracy, P-Precision, R-Recall, RF-Random Forest, DT-Decision Tree, LR-Logistic Regression, SVM-Support Vector Machine, NB-Naïve-Bayes, KNN-K Nearest Neighbour, MLP-Multi-Layer Perceptron. CNN-Convolutional Neural Networks

The highest performance in terms of Accuracy, Precision, Recall, and F1-score are highlighted in bold

categories. The experiments were also carried out with convolutional neural networks and multi-layer perceptron techniques.

Experiments and results

This section discusses the experiments and results. The proposed system is built with a novel method for creating feature vectors based on Huffman coding. A total of 139 features (system call frequencies) from 11,598 data samples provided by CICMaldroid 2020 (Mahdavifar 2020; Mahdavifar et al. 2022; Canadian Institute for Cybersecurity 2020) were used in the system. The proposed feature vector generation technique significantly improves the overall effectiveness of the detection model. Although there are several data scaling techniques, the primary issue for machine learning is selecting the appropriate scaling method. The studies (Ambarwari et al. 2020; Shahriyari 2019) support the impact of data scaling methods on various ML algorithms. As a result, the proposed solution uses standard scaling because it yields better performance with machine learning models. The experimental results are shown in Tables 4 and 5. The corresponding result graphs are depicted in Figs. 4 and 5.

Table 4 shows the results without the proposed feature vector generation. This demonstrates that, with a greater detection accuracy of 0.931, the random forest model performs better. The decision tree, support vector machine, k-nearest neighbor classifiers, multi-layer perceptron and CNN models provide more than 80% of detection accuracies.

The experiment results with the proposed Huffman Encoding-based approach are presented in Table 5. It is evident that the proposed approach yields a greater accuracy of 98.70%. Moreover, it has increased the performance of other classifiers as well. This proves how effectively proposed feature vector generation process works. Huffman encoding supplies an optimised size value for each feature in O(nlogn) times, which increases the detection model's efficacy.

The proposed feature vector generation technique is compared with logarithmic transformation-based feature vector generation (Table 6). As per the results, the log transformation gives the greater accuracy is 93.06% with Random Forest model. It is known that log transformation will reduce the skewness of data by compressing the range of large numbers and extending the range of small numbers. However, it may lead to high memory consumption and increased time complexity. Also, this technique is computationally expensive and it may cause lowering of models' accuracy. Whereas, in the Huffman encoding method, its minimum redundancy code property and higher encoding speed enables it to produce an optimal feature vector, which can improve the performance of the final detection model. The experiments also proved that Huffman-encoding feature vector generation gives better results than logarithmic based feature vector generation. Figure 6 shows the corresponding result graph of logarithmic transformation-based approach.

From Fig. 7, it is clear that the performance of the proposed method is higher than the method without using any feature vector generation and the baseline technique, i.e., logarithmic transformation technique.

As shown in Table 7, the effectiveness of the proposed system is compared to that of a few existing methodologies. The results show that the proposed system outperforms the alternatives.

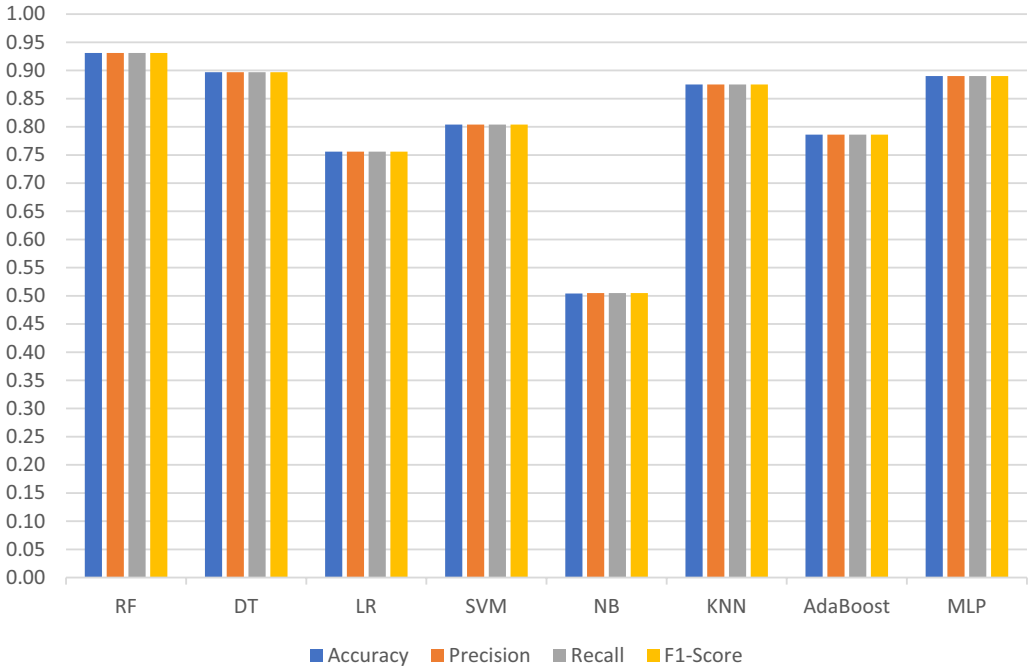

Fig. 4 Results without proposed feature vector generation

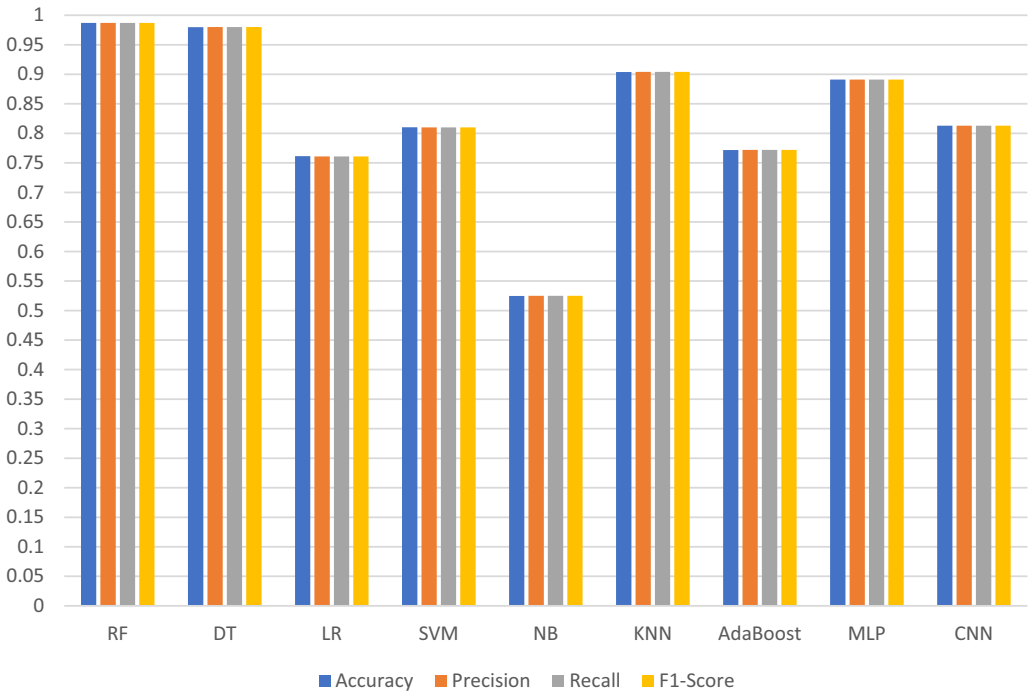

Fig. 5 Results with Huffman encoding-based feature vector generation

Table 6 Results with Logarithmic transformation

Model used	A	P	R	F1-Score
RF	**0.9306**	**0.931**	**0.931**	**0.931**
DT	0.8897	0.890	0.890	0.890
LR	0.8586	0.859	0.859	0.859
SVM	0.8797	0.880	0.880	0.880
NB	0.4715	0.472	0.472	0.472
KNN	0.8926	0.893	0.893	0.893
AdaBoost	0.7862	0.786	0.786	0.786
MLP	0.9254	0.925	0.925	0.925
CNN	0.7762	0.776	0.776	0.776

A-Accuracy, P-Precision, R-Recall, RF-Random Forest, DT-Decision Tree, LR-Logistic Regression, SVM-Support Vector Machine, NB-Naïve-Bayes, KNN-K Nearest Neighbour, MLP-Multi-Layer Perceptron

The highest performance in terms of Accuracy, Precision, Recall, and F1-score are highlighted in bold

Conclusion

This paper presents an Android malware category detection system based on a novel Huffman encoding-based feature vector generation scheme. The proposed model includes phases like data acquisition, data pre-processing, feature selection, and classification. The system call frequencies of 11,598 Android application samples were used as features to design this solution because this dynamic feature helps to recognize the dynamic behavior patterns of malware category. The Huffman encoding technique is employed in the data pre-processing phase to provide the optimum size of system call frequencies used by the applications. Several Machine learning-based experiments are conducted to evaluate the effectiveness of the proposed system. Based on the findings of the experiments, the proposed method using the Random Forest model outperforms other models with a better accuracy of 98.70%. The results were also compared with the performance of logarithmic transformation-based feature vector generation, showing that the proposed approach exhibits better results. Additionally, the model's effectiveness was compared with a few earlier methods, and it was discovered that this work yields better outcomes. This solution relies on a dynamic feature called system calls; thus, in future research studies, the static features like permissions, API calls, intents, etc., should be integrated with the detection solution.

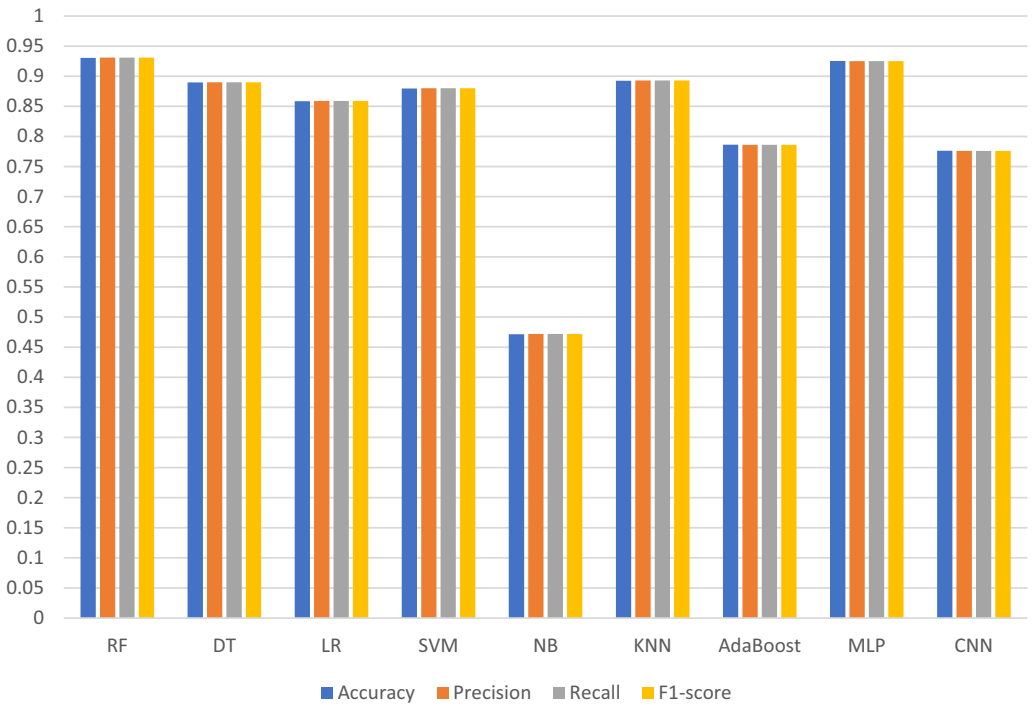

Fig. 6 Results with logarithmic transformation

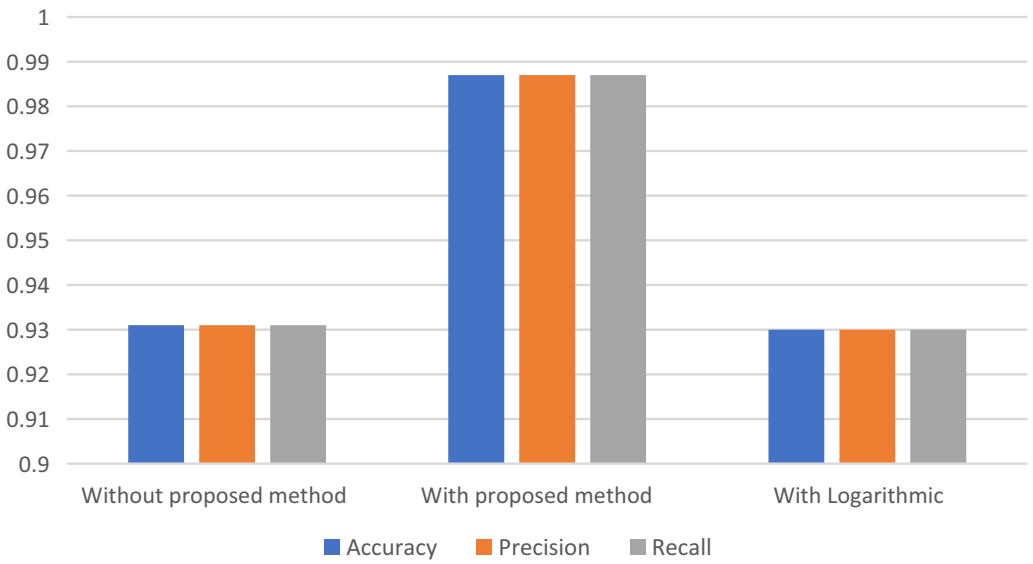

Fig. 7 Result comparison of proposed method

Table 7 Performance comparison of the proposed system with previous methods

Existing methods	Accuracy	Precision	Recall	F1-score	Method used
Martin et al. (2018)	0.761	0.755	0.76	0.755	CNN + Markov chains
Martin et al. (2018)	0.818	0.807	0.818	0.802	RF + Markov chains
Arindaam Roy. et al. (2020)	0.887	0.895	0.819	0.855	SVM + Feature aggregation
Nicheporuk et al. (2020)	0.933	0.938	0.937	0.938	CNN + word2vec technology-based Feature vectorization
Samaneh et al. (2022)	0.982	0.982	0.982	0.982	Semi-supervised DNNs
Hashem A. El Fiky et al. (2021)	0.9689	Not available	0.6646	Not available	RF
Proposed Method	0.9870	0.987	0.987	0.987	RF + Huffman encoding-based Feature Vector Generation

RF-Random Forest, SVM-Support Vector Machine, CNN-Convolutional Neural Networks, DNN-Deep Neural Networks

Acknowledgements
There is no any third person/ organisation to acknowledge

Author contributions
HHRM: Research, Data analysis, Documentation, Reporting, Implementations, Problem formulation, Coding, Testing. Dr. MNS: Supervision, Management, Validation. All authors read and approved the final manuscript.

Funding
The authors declare that the research doesn't used any funding sources for the work. There are no any funding sources to disclose.

Availability of data and materials
The datasets analysed during the current study are available in the site Canadian Institute for Cybersecurity, CICMalDroid 2020 (Canadian Institute for Cybersecurity 2020), https://www.unb.ca/cic/datasets/maldroid-2020.html.

Declarations

Competing interests
The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Received: 24 August 2022 Accepted: 11 January 2023
Published online: 09 March 2023

References

Abderrahmane A, Adnane G, Yacine C, Khireddine G, (2019). Android malware detection based on system calls analysis and CNN classification. In: 2019 IEEE wireless communications and networking conference workshop (WCNCW) (pp 1–6). IEEE

Almahmoud M, Alzubi D, Yaseen Q (2021) ReDroidDet: android malware detection based on recurrent neural network. Procedia Comput Sci 184:841–846. https://doi.org/10.1016/j.procs.2021.03.105
Alswaina F, Elleithy K (2018) Android malware permission-based multi-class classification using extremely randomized trees. IEEE Access. https://doi.org/10.1109/ACCESS.2018.2883975
Alswaina F, Elleithy K (2020) Android malware family classification and analysis: current status and future directions. Electronics 9(6):942
Ambarwari A, Adrian QJ, Herdiyeni Y (2020) Analysis of the effect of data scaling on the performance of the machine learning algorithm for plant identification. J Resti Rekayasa Sist Dan Teknol Inf 4:117–122
Atzeni A, Diaz F, Marcelli A, Sánchez A, Squillero G, Tonda A (2018) Countering android malware: a scalable semi-supervised approach for family-signature generation. IEEE Access. https://doi.org/10.1109/ACCESS.2018.2874502
Bai Y, Xing Z, Ma D, Li X, Feng Z (2021) Comparative analysis of feature representations and machine learning methods in android family classification. Comput Netw 184:107639
Business of Apps: Android Statistics (2022). Android Statistics (2022) - Business of Apps Accessed on 20 July 2022
Canadian Institute for Cybersecurity, CICMalDroid 2020, https://www.unb.ca/cic/datasets/maldroid-2020.html, Accessed on 30 Mar 2022
Contagio Mobile http://contagiominidump.blogspot.com/, Accessed on 30 Mar 2022
CuckooDroid (2020). Cuckoodroid book. Retrieved 2020, from https://cuckoo-droid.readthedocs.io/en/latest/
D'Angelo G, Palmieri F, Robustelli A, Castiglione A (2021) Effective classification of android malware families through dynamic features and neural networks. Connect Sci 33(3):786–801. https://doi.org/10.1080/09540091.2021.1889977
Dhalaria M, Gandotra E (2021) A hybrid approach for android malware detection and family classification. Int J Interact Multimed Artif Intel. https://doi.org/10.9781/ijimai.2020.09.001
Ding C, Luktarhan N, Lu B, Zhang W (2021) A hybrid analysis based approach to android malware family classification. Entropy 23:1009. https://doi.org/10.3390/e23081009
Elayan ON, Mustafa AM (2021) Android malware detection using deep learning. Procedia Comput Sci 184:847–852. https://doi.org/10.1016/j.procs.2021.03.106
Fiky AHE, Shenawy AE, Madkour MA (2021) Android malware category and family detection and identification using machine learning. arXiv preprint https://arxiv.org/abs/2107.01927
Huffman DA (1952) A method for the construction of minimum-redundancy codes. Proc Inst Radio Eng 40(9):1098–1101
Huffman coding, https://en.wikipedia.org/wiki/Huffman_coding, Accessed on 30 Mar 2022
Imtiaz SI, Rehman SU, Javed AR, Jalil Z, Liu X, Alnumay WS (2021) DeepAMD: detection and identification of android malware using high-efficient deep artificial neural network. Future Gener Comput Syst 115:844–856. https://doi.org/10.1016/j.future.2020.10.008
International Conference on Smart Sustainable Intelligent Computing and Applications under ICITETM2020 Android Malware Detection based on Vulnerable Feature Aggregation Arindaam Roya,_, Divjeet Singh Jasa, Gitanjali Jaggia, Kapil Sharmaa
Karbab E, Debbabi M, Derhab A, Mouheb D (2018) MalDozer: automatic framework for android malware detection using deep learning. Digit Investig 24:S48–S59. https://doi.org/10.1016/j.diin.2018.01.007
Kim M, Kim D, Hwang C, Cho S, Han S, Park M (2021) Machine-learning-based android malware family classification using built-in and custom permissions. Appl Sci 11:10244. https://doi.org/10.3390/app112110244
Lee DH (2013) Pseudo-label: the simple and efficient semi-supervised learning method for deep neural networks. Workshop on challenges in representation learning, ICML. 3(2)
Mahdavifar S, Alhadidi D, Ghorbani AA (2022) Effective and efficient hybrid android malware classification using pseudo-label stacked auto-encoder. J Netw Syst Manage 30(1):1–34
Mahindru A, Sangal AL (2021a) MLDroid—framework for Android malware detection using machine learning techniques. Neural Comput Appl 33:5183–5240. https://doi.org/10.1007/s00521-020-05309-4
Mahindru A, Sangal AL (2021b) SemiDroid: a behavioral malware detector based on unsupervised machine learning techniques using feature selection approaches. Int J Mach Learn Cyber 12:1369–1411. https://doi.org/10.1007/s13042-020-01238-9
Mahdavifar S, Kadir AFA, Fatemi R, Alhadidi D, Ghorbani AA (2020) Dynamic android malware category classification using semi-supervised deep learning, In: The 18th IEEE international conference on dependable, autonomic, and secure computing (DASC), 17–24
Martín A, Rodríguez-Fernández V, Camacho D (2018) CANDYMAN: classifying android malware families by modelling dynamic traces with Markov chains. Eng Appl Artif Intell 74:121–133. https://doi.org/10.1016/j.engappai.2018.06.006
Meijin L, Zhiyang F, Junfeng W, Luyu C, Qi Z, Tao Y, Yinwei W, Jiaxuan G (2022) A systematic overview of android malware detection. Appl Artif Intel 36(1):2007327. https://doi.org/10.1080/08839514.2021.2007327
Moffat A (2019) Huffman coding. ACM Comput Surv (CSUR) 52(4):1–35
Nicheporuk A, Savenko O, Nicheporuk A, Nicheporuk Y (2020) An android malware detection method based on CNN mixed-data model CEUR Workshop Proceedings Kharkiv, Ukraine. 2732:198–213
Oyama Y, Giang TTD, Chubachi Y, Shinagawa T, Kato K (2012) Detecting malware signatures in a thin hypervisor, In: Proceedings of the 27th Annual ACM symposium on applied computing, SAC 12, ACM, New York, NY, USA, pp 1807–1814, https://doi.org/10.1145/2245276.2232070
Pei X, Long Y, Tian S (2020) AMalNet: a deep learning framework based on graph convolutional networks for malware detection. Comput Secur 93:101792. https://doi.org/10.1016/j.cose.2020.101792
Portokalidis G, Slowinska A, Bos Argos H (2006) An emulator for fingerprinting zero-day attacks for advertised honeypots with automatic signature generation, In: Proceedings of the 1st ACM SIGOPS/EuroSys European Conference on Computer Systems 2006, EuroSys '06, ACM, New York, NY, USA, pp 15–27, https://doi.org/10.1145/1217935.1217938
Shahriyari L (2019) Effect of normalization methods on the performance of supervised learning algorithms applied to HTSeq-FPKM- UQ data sets: 7SK RNA expression as a predictor of survival in patients with colon adenocarcinoma. Briefings Bioinform 20:985–994
Shao K, Xiong Q, Cai Z (2021) FB2Droid: a novel malware family-based bagging algorithm for android malware detection. Secur Commun Netw
Statista: Share of Android OS of global smartphone shipments from 1st quarter 2011 to 2nd quarter 2018* (2022) Android global phone market share 2018 | Statista Accessed on 21 July 2022
Taheri L, Kadir AFA, Lashkari AH (2019) Extensible android malware detection and family classification using network-flows and API-calls. In: 2019 International carnahan conference on security technology (ICCST) (pp 1–8). IEEE
Tchakounté F, Ngassi RCN, Kamla VC et al (2021) LimonDroid: a system coupling three signature-based schemes for profiling Android malware. Iran J Comput Sci 4:95–114. https://doi.org/10.1007/s42044-020-00068-w
Virus Total (2022) https://www.virustotal.com/gui/home/upload, Accessed on 30 Mar 2022
Wressnegger C, Freeman K, Yamaguchi F, Rieck K (2017) Automatically inferring malware signatures for anti-virus assisted attacks. In: Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security, ASIA CCS '17, ACM, New York, NY, USA, pp 587–598, https://doi.org/10.1145/3052973.3053002
Zhang L, Thing VL, Cheng Y (2019) A scalable and extensible framework for android malware detection and family attribution. Comput Secur 80:120–133
Zhou H, Yang X, Pan H, Guo W (2020) An android malware detection approach based on SIMGRU. IEEE Access 8:148404–148410. https://doi.org/10.1109/ACCESS.2020.3007571
Zhu H, Li Y, Li R, Li J, You Z, Song H (2021) SEDMDroid: an enhanced stacking ensemble framework for android malware detection. IEEE Trans Netw Sci Eng 8(2):984–994. https://doi.org/10.1109/TNSE.2020.2996379

Publisher's Note