DNNdroid: Android Malware Detection Framework Based on Federated Learning and Edge Computing

Arvind Mahindru[1](✉) and Himani Arora[2]

[1] Department of Computer Science and Applications, DAV University, Jalandhar, India
er.arvindmahindru@gmail.com
[2] Department of Mathematics, Guru Nanak Dev University, Amritsar, India

Abstract. The fact that apps are available for free via Android's official store has helped the platform become more popular. The functionality of Android apps is reliant on permissions. Due to these permissions, cybercriminals developed malware-infected apps for smartphone users. The main fault lies in the permission model of Android. To address this issue, a framework entitled "DNNdroid" is proposed that work on the principle of federated learning. Information related to newly installed apps is stored on the user's device only and this information is not revealed to the developer. In the meantime, input from all the users is collected simultaneously to train the model with a federated learning process, so that a better classification model is developed. The main challenge in this is that a user is not able to identify whether an app is malware-infected or not. The experiment result reveals that the cloud server has an F1 score of 97.8% having a recall rate of client than 0.95 false positive rates using 1,00,000 unique Android apps with 500 plus users and 50 rounds of the federation. Further, an experiment is performed by using frameworks available in the literature and different anti-virus scanners.

Keywords: Android apps · Smartphones · Machine learning · Deep neural network · Security

1 Introduction

According to the global data report[1], Android has captured 71.54% market share. The main reason for its popularity is the availability of free apps in its official play store. Cybercriminals are taking advantage of this and developing malware-infected apps on a daily basis for smartphone users. In the literature [5,16–18], researchers and academicians proposed different malware detection frameworks that work on machine learning techniques and achieved success too. Often, developing an accurate malware detection model with classification machine learning algorithms is dependent upon the extensive collection of datasets. But it has

[1] https://gs.statcounter.com/os-market-share/mobile/worldwide.

https://doi.org/10.1007/978-3-031-23095-0_7

a limitation, it affects the privacy of smartphone users [23]. To address these issues, there is a need for decentralized entry information which also respects the user's privacy and will not expose to third parties too.

In the literature [14,19], academicians and researchers proposed three different machine learning solutions for malware detection i.e., client-based, cloud-based, and hybrid (combination of cloud-based and client-based) techniques. In the client-based approach, collected data is protected due to the machine learning model being developed locally on the host computer but the process is time-consuming and has the highest value of false positives. In the cloud-based approach, this process is reversed of it. It is developed by using a large set of features and reveals which app is installed by the users too. Last, in the hybrid model, Android apps that are malware-infected are sent to the cloud for further analysis. This type of solution has a high number of false positives and reveals users' private data to the cloud.

In identifying malware from Android devices, machine learning algorithms are without a doubt incredibly effective. But, to develop effective malware detection model, a large amount of information is required. In the literature [25], it was observed that large amount of features are available at the central place for training and testing the model. Addition to it, it has seen that developed model memorize and disclose information related to the dataset. To address this issue, we consider the key question that is to be answer in this study, i.e. *How can we create a decentralized, privacy-preserving classifier for Android malware?*

In this study, we proposed DNNdroid - a model that is based on classification techniques and uses the principle of federated learning and respecting the user's privacy. The proposed model collects the features from the user's smartphone without prior knowledge that an app was installed from its official play store or any other promised repositories. The proposed framework reduces the dependency of users on cloud-based technique and also benefit them in term of privacy.

In the literature [1,2,6,12,21], state-of-the-art federated learning techniques were discussed by researchers and academicians. In which, smartphone users test their data locally by using a supervised machine learning algorithm and the resultant performance is updated to the cloud for the betterment of the model. Our proposed model enhances the existing work by incorporating the principle of deep learning at the time of training the model. Further, we evaluate our proposed model by using 1,00,000 unique Android apps out of which 75,000 are benign and 25,000 are malware-infected apps with 500 plus users and 108 rounds of the federation. Additionally, we contrasted our approach with pre-existing frameworks found in the literature and several anti-virus scanners sold today.

The novel and unique contributions of this study are as under:

- To the best of our knowledge, this is the first research paper, which trained with the help of dynamic features of Android apps and prevent the users privacy too.
- In this study, we also demonstrate the effectiveness of our proposed model against malware-infected apps.

The rest of the paper is organized as follows. In Sect. 2, we discuss the related work done in the field of Android malware detection using federated learning. Section 3, described the collection of datasets from different promised repositories. The machine learning technique implemented in our proposed framework is discussed in Sect. 4. Section 5 describes the architecture of our proposed framework. Evaluating the proposed framework is discussed in Sect. 6. In Sect. 7, we compare our proposed framework with the existing framework and the distinct anti-virus available in the market. The experimental finding is discussed in Sect. 7. Section 8 discusses the conclusion and the future scope of this study.

2 Related Work

Hsu et al. [9] proposed a malware detection for Android named as privacy-preserving federated learning (PPFL). The proposed model is trained by using SVM as a base classifier. They developed their model by trained them using static analysis. Experiment result reveals that proposed model achieved higher detection rate as compared to decentralized models. Empirical results also reveals that if number of clients increases the accuracy is also increases. Gálvez et al. [6] proposed a malware detection model named as LiM that work on the principles of semi-supervised machine learning technique and federated learning. Experiment was performed on 50,000 Android apps having 200 users and 50 rounds of federation. Taheri et al. [26] proposed malware detection model entitled FEd-IIoT for detecting malware in IIoT. The results of the experiment corroborate the high accuracy rates of our attack and defence algorithms and demonstrate how the A3GAN defensive strategy protects the robustness of data privacy for Android mobile users and is around 8% more accurate than current state-of-the-art solutions.

3 Datasets

In this study, we collect Android application packages (.apk) from Google play store[2], AppChina[3], Android[4] and Mumayi[5]. Malware-infected apps were collected from AndroMalShare[6] and Malgenomeproject [29]. Table 1 represent the collected Android apps.

Feature Dataset. Extraction of features are done as per the study [15]. 1844 distinct features are extracted from collected Android apps. Features play an important role to train the classification model. In the literature [11,20], different feature selection techniques were proposed by researchers and academics. In this work, we implement chi-square test to select significant features that helps to train the model.

[2] https://play.google.com/store?hl=en.
[3] http://m.appchina.com/.
[4] https://android.d.cn/.
[5] http://www.mumayi.com/.
[6] http://sanddroid.xjtu.edu.cn:8080/.

Table 1. Collected Android apps.

Category	Normal	Trojan	Backdoor	Worms	Botnet	Spyware
Books and reference	12205	100	16	52	120	150
Business	1308	110	120	130	22	14
Casual	11271	3250	79	66	160	130
Communication	1414	2500	50	420	33	23
Entertainment	1222	5000	500	400	102	40
Health and fitness	1551	98	85	25	120	160
Lifestyle	10650	135	150	150	195	190
News and magazines	1064	500	42	500	200	22
Social	2159	100	240	300	450	112
Weather	11841	2	200	300	200	10

4 Machine Learning Technique

Deep Neural Network (DNN) is implemented to train the model in cloud-based architecture i.e., base learner in our study. In the literature, authors proposed two distinct methods to develop model using DNN i.e., Deep Belief Networks (DBN) and Convolutional neural networks (CNN). In the current study, we decide to build our deep learning model using DBN architecture. The deep learning method's architecture is shown in Fig. 1. It consists of two stages: supervised back propagation in the first and unsupervised pre-training in the second. Restricted Boltzmann Machines (RBM) and a deep neural network are used to train the model in the initial stages of model construction. The model is built

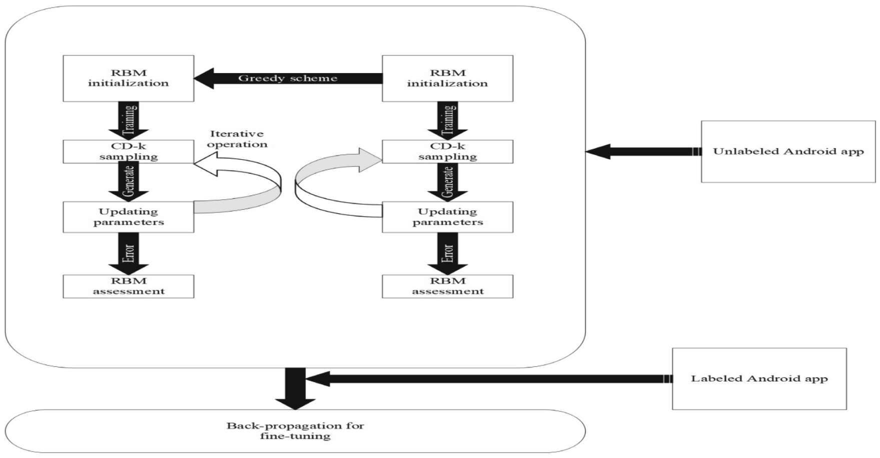


Fig. 1. Architecture of deep neural network.

using an iterative procedure in the training phase using unlabeled Android apps. Pre-trained DBN is adjusted using labelled Android apps in a supervised way during the back-propagation step. An Android app is used in both stages of the training process for a model created using the deep learning technique.

5 Proposed Framework Architecture

Federated learning is implemented based on a decentralized approach to training the model. Clients implement the process locally and the outcome is shared with the service provider. The main success of federated learning is dependent upon the labeled dataset which can be used to train the model. But it has one limitation, smartphone users do not know what to label malicious or benign. To overcome this issue, in our study we implement supervised learning at the cloud-based structure i.e., labeled dataset, and unsupervised learning at the client-based structure i.e., unlabelled dataset.

In the proposed framework, the federation of learning has happened in the cloud database and the client estimates the unlabeled dataset at the time of testing. In addition, this cloud server collects all the data from clients and aggregates them, and presents the weight. Figure 2 demonstrates the architecture of the proposed work. The following steps are taken to train and evaluate the model.

1. **Server Side:** First of all, labeled data is given to the server to train it as a base classifier and send unlabeled data to assess the weight from it.
2. **Client Side:** Client receive baseline classifier and base learner.
3. **Weight Gain:** Client gained the estimated weights from installed apps.

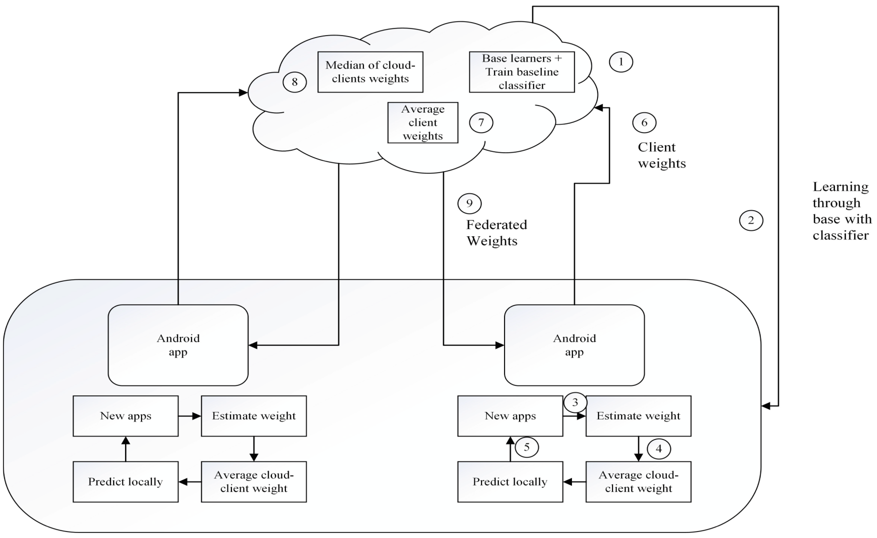

Median of cloud-clients weights
Base learners + Train baseline classifier
1
8
Average client weights
7
6
Client weights
9
Federated Weights
2
Learning through base with classifier
Android app
Android app
New apps
Estimate weight
Predict locally
Average cloud-client weight
3
4
5

Fig. 2. Proposed framework i.e., DNNDroid.

4. **Calculate Estimate Weight:** Client calculates the average weight.
5. **Predict at the Client Side:** Client classifies the installed apps.
6. **Complete the process:** Client computes the aggregate weight and uploads them to the cloud for further processing.
7. **Aggregate at the cloud:** Cloud collects all the weight and averages them.
8. **Median client-cloud weight:** At last the median weight of both client and cloud are computed and used for further processing.

6 Evaluation of Proposed Framework

To evaluate the proposed framework, we set up a server with 500 users iterating over 100 different federation rounds. The client model is run parallelly on Intel Core i7 machine having 16 GB RAM. In this study, we consider two different parameters to evaluate our proposed model i.e., Accuracy and F-measure. Table 2 shows the confusion matrix for determining if an app is malware-infected or benign.

Table 2. Confusion matrix consider in this study. (*.apk*)

	Benign	Malware
Benign	Benign->Benign (TP)	Benign->Malware (FP)
Malware	Malware->Benign (FN)	Malware->Malware (TN)

Following terminology are used in this study for evaluate the proposed framework.

- Recall: Recall measures the number of precise class predictions generated from all of the positive examples in the dataset.

$$Recall = \frac{a}{a+c}, \tag{1}$$

  where $a = N_{Malware \rightarrow Malware}$,
  $b = N_{Benign \rightarrow Malware}$,
  $c = N_{Malware \rightarrow Benign}$

- Precision: Precision is the percentage of predicted members of a positive class that really belong to that class.

$$Precision = \frac{a}{a+b}. \tag{2}$$

Accuracy: Accuracy is computed as mentioned in [14]:

$$Accuracy = \frac{a + d}{N_{classes}}, \tag{3}$$

where $N_{classes} = a + b + c + d$,
$d = N_{Benign \rightarrow Benign}$

F-measure: F-measure is computed as mentioned in [14]:

$$\begin{aligned} F - measure =& \frac{2 * Recall * Precision}{Recall + Precision} \\ =& \frac{2 * a}{2 * a + b + c} \end{aligned} \tag{4}$$

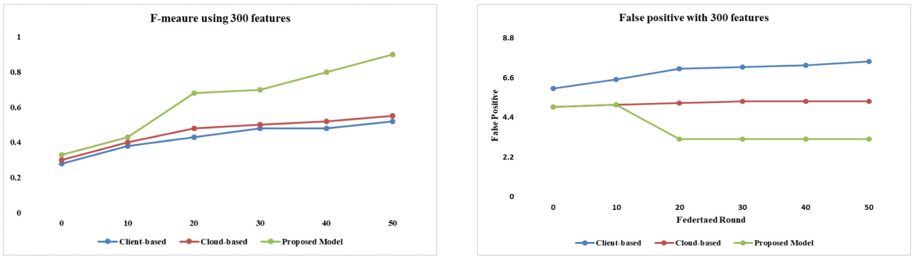
Table 3 shows the calculated value of accuracy and F-measure using above mentioned equations features selected by using chi-square analysis. From empirical study, it can be observed that by using 300 unique features we gain the optimal value in terms of detection rate. Figure 3, demonstrate the computed value of the F-measure and False positive rate having 40 round of federation.

Table 3. Calculated Accuracy and F-measure by using first 50, 100, 150, 200 and 300 features having 50 rounds of federation learning.

ID	Accuracy					F-measure				
	50	100	150	200	300	50	100	150	200	300
C1	**81.4**	80.0	**81.4**	80.6	**81.4**	**0.8**	0.73	**0.8**	0.72	**0.8**
C2	82.4	83	83.4	81.9	**93.8**	0.77	0.79	0.82	0.83	**0.94**
C3	82.4	83	83.4	83	**94.9**	0.82	0.81	0.82	0.83	**0.94**
C4	80.4	83	83.4	83.8	**93.8**	0.83	0.80	0.81	0.82	**0.93**
C5	80.4	82	83.4	84.8	**94.9**	0.82	0.83	0.83	0.82	**0.94**
C6	83.4	84	83.4	84.8	**95.2**	0.83	0.84	0.82	0.84	**0.95**
C7	80.4	83	83.4	83.8	**93.8**	0.83	0.80	0.81	0.82	**0.93**
C8	83.4	82	84.4	83	**94**	0.73	0.75	0.80	0.82	**0.94**
C9	80.4	83	83.4	83.8	**93.8**	0.83	0.80	0.81	0.82	**0.93**
C10	80.4	83	83.4	83.8	**93.8**	0.83	0.80	0.81	0.82	**0.93**

Based on Table 3 and Fig. 3, we are having the following observations:

– It can be inferred that how an optimal number of features are required to train the model.
– It can reveal that increasing the value of the federated learning model it is a directly paid impact on the detection rate of malware-infected apps.
– By increasing the value of the federated model, it can also be paid to impact the value of the false positive rate.



(a) F-measure (b) False Positive Rate

Fig. 3. F-measure and False positive rate after federation round.

7 Comparison of Proposed Framework

In this study, we select two different methods to validate our proposed work which are described below:

7.1 Comparison on the Basis of Framework Available in the Literature

To examine whether our proposed framework is equivalent to a previously developed framework or not, we compare our proposed framework with ten distinct previously developed frameworks available in the literature. To perform this, we consider Drebin dataset [3] in our study. Table 4 shows the result of our empirical analysis.

7.2 Comparison of the Proposed Framework with Different Anti-virus Scanners

In this study, we evaluate the available free antivirus scanners on the market with our suggested framework for detecting malware. In this study, the Drebin dataset is used to provide empirical results. Table 5 compares various antivirus scanners using the structure we've suggested.

8 Experimental Findings

Based on the experimental outcomes, the following are the experimental findings of this research article.

- Tables 4 and 5 provide evidence that the suggested framework is capable of identifying malware in real-world apps.
- Based on empirical findings, it can be concluded that the suggested methodology can identify malware-infected apps more quickly than other anti-virus scanners on the market.

Table 4. Comparing the proposed framework to existing frameworks or methods.

Framework/Approach	Detection rate
Andromaly [24]	78.9%
DroidDet [31]	76.3%
AndroSimilar [5]	78.2%
Aurasium [28]	79.1%
Andrubis [13]	82.5%
TaintDroid [4]	88.9%
Paranoid Android [22]	89.9%
MalDozer [10]	80.5%
HinDroid [8]	86.3%
HEMD [30]	87.9%
MalInsight [7]	90.2%
Wei Wang [27]	93.8%
Proposed framework	96.3%

Table 5. Comparative analysis using various antivirus scanners.

Name of the anti-virus	Detection rate (in %)	Speed to detect malware in sec
Ikarus	81.68	68
McAfee	82.9	38
AVG	91.2	30
ESET NOD32	93.9	29
Proposed framework	98.5	12

9 Conclusion

Based on empirical studies, it can be concluded that our suggested framework has an accuracy of 98.7% and can identify malware-infected apps with 500 unique attributes and 40 different federation rounds. Additionally, experimental results show that our suggested framework is more accurate than other anti-virus scanners and proposed frameworks in the literature. Further, the work will be extended by implementing distinct feature selection approaches and soft computing techniques.

References

1. Alyamani, H.J.: Cyber security for federated learning environment using AI technique. Expert Syst. e13080 (2022)
2. Arisdakessian, S., Wahab, O.A., Mourad, A., Otrok, H., Guizani, M.: A survey on IoT intrusion detection: federated learning, game theory, social psychology and explainable AI as future directions. IEEE Internet Things J. (2022). https://doi.org/10.1109/JIOT.2022.3203249
3. Arp, D., Spreitzenbarth, M., Hubner, M., Gascon, H., Rieck, K., Siemens, C.: Drebin: effective and explainable detection of android malware in your pocket. In: NDSS, vol. 14, pp. 23–26 (2014)
4. Enck, W., et al.: TaintDroid: an information-flow tracking system for realtime privacy monitoring on smartphones. ACM Trans. Comput. Syst. (TOCS) **32**(2), 1–29 (2014)
5. Faruki, P., Ganmoor, V., Laxmi, V., Gaur, M.S., Bharmal, A.: AndroSimilar: robust statistical feature signature for android malware detection. In: Proceedings of the 6th International Conference on Security of Information and Networks, pp. 152–159 (2013)
6. Gálvez, R., Moonsamy, V., Diaz, C.: Less is more: a privacy-respecting android malware classifier using federated learning. arXiv preprint arXiv:2007.08319 (2020)
7. Han, W., Xue, J., Wang, Y., Liu, Z., Kong, Z.: MalInsight: a systematic profiling based malware detection framework. J. Netw. Comput. Appl. **125**, 236–250 (2019)
8. Hou, S., Ye, Y., Song, Y., Abdulhayoglu, M.: HinDroid: an intelligent android malware detection system based on structured heterogeneous information network. In: Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 1507–1515 (2017)
9. Hsu, R.H., et al.: A privacy-preserving federated learning system for android malware detection based on edge computing. In: 2020 15th Asia Joint Conference on Information Security (AsiaJCIS), pp. 128–136. IEEE (2020)
10. Karbab, E.B., Debbabi, M., Derhab, A., Mouheb, D.: MalDozer: automatic framework for android malware detection using deep learning. Digital Invest. **24**, S48–S59 (2018)
11. Kumar, L., Misra, S., Rath, S.K.: An empirical analysis of the effectiveness of software metrics and fault prediction model for identifying faulty classes. Comput. Stand. Interfaces **53**, 1–32 (2017)
12. Lin, K.Y., Huang, W.R.: Using federated learning on malware classification. In: 2020 22nd International Conference on Advanced Communication Technology (ICACT), pp. 585–589. IEEE (2020)

13. Lindorfer, M., Neugschwandtner, M., Weichselbaum, L., Fratantonio, Y., Veen, V.V.D., Platzer, C.: ANDRUBIS-1,000,000 apps later: a view on current android malware behaviors. In: 2014 Third International Workshop on Building Analysis Datasets and Gathering Experience Returns for Security (BADGERS), pp. 3–17. IEEE (2014)
14. Mahindru, A., Sangal, A.: MLDroid-framework for android malware detection using machine learning techniques. Neural Comput. Appl. **33**, 5183–5240 (2020)
15. Mahindru, A., Sangal, A.: PARUDroid: validation of android malware detection dataset. J. Cybersecurity Inf. Manage. **3**(02), 42–52 (2020)
16. Mahindru, A., Sangal, A.L.: PerbDroid: effective malware detection model developed using machine learning classification techniques. In: Singh, J., Bilgaiyan, S., Mishra, B.S.P., Dehuri, S. (eds.) A Journey Towards Bio-inspired Techniques in Software Engineering. ISRL, vol. 185, pp. 103–139. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-40928-9_7
17. Mahindru, A., Sangal, A.L.: SOMDROID: android malware detection by artificial neural network trained using unsupervised learning. Evol. Intell. **15**(1), 407–437 (2020)
18. Mahindru, A., Sangal, A.: HybriDroid: an empirical analysis on effective malware detection model developed using ensemble methods. J. Supercomput. **77**(8), 8209–8251 (2021)
19. Mariconti, E., Onwuzurike, L., Andriotis, P., Cristofaro, E.D., Ross, G., Stringhini, G.: MamaDroid: detecting android malware by building Markov chains of behavioral models. arXiv preprint arXiv:1612.04433 (2016)
20. Martín, A., Menéndez, H.D., Camacho, D.: MOCDroid: multi-objective evolutionary classifier for android malware detection. Soft Comput. **21**(24), 7405–7415 (2017)
21. Pei, X., Deng, X., Tian, S., Zhang, L., Xue, K.: A knowledge transfer-based semi-supervised federated learning for IoT malware detection. IEEE Trans. Dependable Secure Comput. **21**, 7405–7415 (2022)
22. Portokalidis, G., Homburg, P., Anagnostakis, K., Bos, H.: Paranoid android: versatile protection for smartphones. In: Proceedings of the 26th Annual Computer Security Applications Conference, pp. 347–356 (2010)
23. Saracino, A., Sgandurra, D., Dini, G., Martinelli, F.: MADAM: effective and efficient behavior-based android malware detection and prevention. IEEE Trans. Dependable Secure Comput. **15**(1), 83–97 (2016)
24. Shabtai, A., Kanonov, U., Elovici, Y., Glezer, C., Weiss, Y.: "Andromaly": a behavioral malware detection framework for android devices. J. Intell. Inf. Syst. **38**(1), 161–190 (2012)
25. Song, C., Ristenpart, T., Shmatikov, V.: Machine learning models that remember too much. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, pp. 587–601 (2017)
26. Taheri, R., Shojafar, M., Alazab, M., Tafazolli, R.: FED-IIoT: a robust federated malware detection architecture in industrial IoT. IEEE Trans. Ind. Inf. **17**(12), 8442–8452 (2020)
27. Wang, W., Zhao, M., Wang, J.: Effective android malware detection with a hybrid model based on deep autoencoder and convolutional neural network. J. Ambient Intell. Humanized Comput. **10**(8), 3035–3043 (2019)
28. Xu, R., Saidi, H., Anderson, R.: Aurasium: practical policy enforcement for android applications. In: Presented as part of the 21st USENIX Security Symposium USENIX Security 2012, pp. 539–552 (2012)

29. Zhou, Y., Jiang, X.: Dissecting android malware: characterization and evolution. In: 2012 IEEE Symposium on Security and Privacy, pp. 95–109. IEEE (2012)
30. Zhu, H.J., Jiang, T.H., Ma, B., You, Z.H., Shi, W.L., Cheng, L.: HEMD: a highly efficient random forest-based malware detection framework for android. Neural Comput. Appl. **30**(11), 3353–3361 (2018)
31. Zhu, H.J., You, Z.H., Zhu, Z.X., Shi, W.L., Chen, X., Cheng, L.: DroidDet: effective and robust detection of android malware using static analysis along with rotation forest model. Neurocomputing **272**, 638–646 (2018)