ICIC Express Letters
Part B: Applications
Volume **14**, Number **7**, July **2023**

 ISSN 2185-2766
pp. **727–734**

ENSEMBLE LEARNING FOR PERMISSION-BASED MOBILE MALWARE DETECTION IN ANDROID

Gracia Angelica Nadaputri and Dennis Gunawan*

Department of Informatics
Universitas Multimedia Nusantara
Jl. Boulevard, Gading Serpong, Kelapa Dua, Kab. Tangerang 15811, Indonesia
gracia.nadaputri@student.umn.ac.id; *Corresponding author: dennis.gunawan@umn.ac.id

Received November 2022; accepted February 2023

Abstract. *Mobile devices have been an essential part of society and with the ever-growing market share, the potential for harm has also increased. With that being said, the implementation of machine learning to aid in preventing these threats is also on the rise. One approach is to use an ensemble learning method in which smaller and lighter models, also known as base learners, are put together to predict the same problem with higher accuracy. The models used in this exploration include Decision Trees, Naïve Bayes Classifiers, and Logistic Regression. Two ensemble methods were tested when put together or put in groups: bagging and stacking. Multiple combinations were tested with these models and methods on a dataset which consists of 50,000 data of applications, each with 1,436 features. The most successful model, a Bagged Decision Tree Ensemble, produced an accuracy of 90.67%, AUC score of 84.44%, and precision value of 91.10%.*
Keywords: Android permission-based malware detection, Decision Trees, Ensemble learning, Logistic Regression, Naïve Bayes Classifier

1. **Introduction.** Android has been a leader in the mobile phone OS market with over two and a half billion devices running it [1]. With such a large reach, the importance of keeping the Android devices secured becomes a quintessential concern for all as these systems hold important assets such as personal information and credential data [2]. An aspect of Android systems that enhances its popularity is its open-sourced nature which can help in innovative development, but it also allows customization of the overall system [3]. As a result, malicious perpetrators can develop and implement security vulnerabilities to be deployed in a potential user's device [4]. A possible medium that perpetrators can use to distribute their customized vulnerabilities is through mobile applications published in the application market. Consequently, the open-sourced nature of Android allows users to download and install these applications from any given source: official ones under Google and unofficial third-party sources [5].

Over the years, Android as an operating system has attempted to reduce the amount of malware, or simply complicated the process of malware being installed in an Android system [6]. These include Bouncer, the in-house malware detection system implemented on the Google Play Store through the Android permission system [7]. Bouncer was proven to be outdated and unsatisfactory due to its lack of sensitivity in detecting malware [6], while the detection system by the Google App Store lacks protection as it only detects malicious intent through analysis of the application metadata as the application is uploaded into the Play Store [8].

With the constant evolution of malware, the implementation of machine learning models in detecting malicious software has been gaining traction [2], specifically those that process a variety of features within the Android system. In detecting malware, a static analysis

DOI: 10.24507/icicelb.14.07.727

examines an application's code and determines its intent without executing it [10], while a dynamic analysis monitors an application in an executed state. Static analysis has a bigger advantage in terms of detecting malware as it serves as a preemptive attempt to halt a possible attack. On the other hand, dynamic approaches require more resources as it needs to run the application in question [11].

Accordingly, this paper will take a static approach to the detection of malicious Android applications. In static approaches, individual models are implemented on extracted features from the application code and are proven to be effective, such as DroidMat and DREBIN [12-14]. As one of those features, permissions play an important role. Strict permissions have also been analyzed and proven to be effective, seen through their high accuracy values [15,16]. As in [15], Random Forest algorithm produced the highest accuracy of 81% on a dataset of 10,000 data points, compared to Support Vector Machine (SVM), Gaussian Naïve Bayes, and K-Means. Choosing the right models to implement in a detection system is crucial, as they need to be the most accurate and effective option [17]. Therefore, a comparison of the features and different individual models that could be used to detect a possible malware was made [18].

Based on the comparisons of models and methodologies, the models that resulted in the highest accuracy using static analysis are Naïve Bayes, Decision Trees, and Logistic Regression [16,19]. As these models work well on their own, the question that is posed consequently is how it can be improved. One approach is to implement Ensemble Learning Systems using these proven models [20,21]. These three algorithms also yielded a 97 to 99 percent detection rate [22] which further the implication that the implementation of an ensemble learning system would prove to be more effective than the single model traditional approach. However, these algorithms were implemented on an older Android system. Hence, this study will implement the same three algorithms on an updated Android version. Different ensemble methodologies were compared and the result showed that the accuracy of the models was elevated to 0.970 and 0.893 through the Bagging method [23] and Stacking method [24], respectively.

After combining the outcomes of the studies gathered, this paper further implements the Bagging and Stacking ensemble learning systems using the three base models: Naïve Bayes, Decision Trees, and Logistic Regression. These models will be used to detect malicious applications within the Android Mobile System, based on the permissions an application requires using a recent dataset with 50,000 application data. The increased size of the dataset compared to previous studies will be used to further confirm and analyze the effectivity of implementing ensemble learning systems in malware detection within Android mobile systems.

The remainder of the paper is composed as follows. First, Section 2 describes the methodology used in this research. In Section 3, the results of the experiment are discussed. Finally, Section 4 concludes this research along with the future works.

2. **Methodology.**

2.1. **Data collection.**

The dataset used in this exploration is a dataset titled *Android Permissions Dataset* [25]. This dataset holds data on 50,000 applications coming from the Android Play Store and third parties. Each data entry holds information on the application and the permissions required before installation or during run time. In total, each data entry holds 1,436 features or possible permissions. Initially, the dataset separates the sources of the applications and holds a separate list of applications that are considered malware. With that, data labeling was carried out based on the provided data.

2.2. **Research design.**

The exploration for this research was conducted following the application model depicted in Figure 1. Firstly, the data collection and preprocessing phase was carried out. In this phase, the data was labeled and exported into data frames

for ease of processing. From then on, the non-binary values were removed and all that was left became the features columns and the labels produced. Moving to the next phase, each individual model was trained, tested, and evaluated. In this stage, the data was split into training and testing data. Following the split, the data was then fit into the individual models, and then trained and tested. After each iteration in which the model was tested, evaluation metrics were calculated. This process was repeated based on the previously set scope. Once the best parameters were gathered, they were used in assembling the ensemble models. The final phase was the ensemble model training, testing, and evaluation. The steps needed for the ensemble models were similar to the individual models. However, rather than tuning the parameters, different combinations of ensembles were created based on the best parameters from the previous phase. The effectiveness of the ensemble models was assessed by the end of the last process through several evaluation metrics.

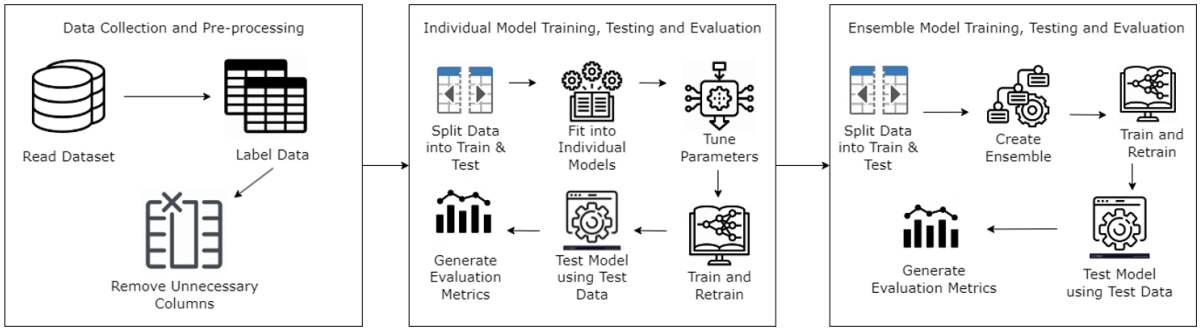

FIGURE 1. Application model of ensemble learning model building

2.3. **Experimentation.** As previously mentioned, multiple experiments were carried out by tuning the parameters of the individual models and through the combinations of the ensemble models. In terms of the individual models, the parameter tuning values are shown in Table 1.

TABLE 1. Individual model parameter testing values

Individual model testing	
Scenario	**Parameters to be tested and range of values**
Decision Trees	Max Depth (1-100), Random State (1-100), Criterion (Gini, Entropy)
Naïve Bayes	Alpha Value (1-100), Random State (1-100), Type (BernoulliNB, GaussianNB)
Logistic Regression	Random State (1-100), Solver (liblinear, lbfgs, saga)

Similarly, there are testing parameters and conditions for the ensemble models. The two main techniques used for the ensemble methods include bagging and stacking. In terms of bagging, the experimentation to see the effect of increasing the number of base learners on the accuracy of the ensemble model was conducted. The number of base learners with the best accuracies was then used in the stacking method, to investigate whether the stacking method could be used to further improve the accuracy of the model. Additionally, stacking the same amount of each base learner into a bigger model was carried out. The values tested on each method are summarized in Table 2.

In both testing the individual models and the ensemble models, only one parameter was changed in each test. This enables the gathering of the best value for the final concluding model that has the highest evaluation metric.

Table 2. Ensemble model testing values

Ensemble model testing	
Classifiers	**Values to be tested**
Bagging classifier	Number of Base Learners (1-100)
Stacking classifier	Set Number (results from Bagging), Stacking Same Amount (1-100)

3. **Results and Discussion.** The classification models are mainly evaluated on the three metrics as follows.

1) Accuracy describes the correctness of a prediction produced by a model and summarized in Equation (1).

$$Accuracy = \frac{True\ Positive\ (TP) + True\ Negative\ (TN)}{TP + TN + False\ Negative\ (FN) + False\ Positive\ (FP)} = \frac{TP + TN}{P + N} \quad (1)$$

2) Precision on the other hand is the count of correctly predicted positive data points produced by the results of the model. The formula to calculate this can be mathematically summarized as shown in Equation (2).

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

3) AUC value is based on the ROC curve, also known as the Receiver Operating Characteristic curve, and is drawn to depict the relationship between the true positive rate, and the false positive rate within a threshold. The area under the ROC curve sums up the ability of a model in ranking predictions. A higher AUC value indicates that the model will have a higher probability in properly detecting a prediction.

With the metrics calculated, when choosing models to form into an ensemble, the accuracy scores were prioritized. Based on the iterations done to get the best values, it can be concluded that the best accuracy for each individual model was obtained through the parameters displayed in Table 3.

Table 3. Parameter configuration that resulted in the highest accuracy

Best values of individual models				
Model	**Parameters**	**Acc**	**Prec**	**AUC**
Decision Trees	Max Depth = 56; Random State = 44; Criterion = Entropy	0.8796	0.7890	0.7544
Naïve Bayes	Alpha = 82; Type = Bernoulli	0.8321	0.5514	0.6282
Logistic Regression	Random State = 0; Solver = liblinear	0.8596	0.7374	0.6558

When assembling a bagging group, the previous parameters were used to set the base learners. Through the process of bagging, Table 4 shows the number of base learners needed to achieve the highest evaluation metrics. The accuracy changes through the increase of the number of base learners, as shown in Figure 2.

Table 4. Bagging parameter values with the highest evaluation metrics

Best values from bagging				
Base learner	**# of base learners**	**Accuracy**	**Precision**	**AUC score**
Decision Trees	61	0.9067	0.9110	0.8444
Naïve Bayes	9	0.8337	0.5488	0.6274
Logistic Regression	3	0.8608	0.7734	0.6624

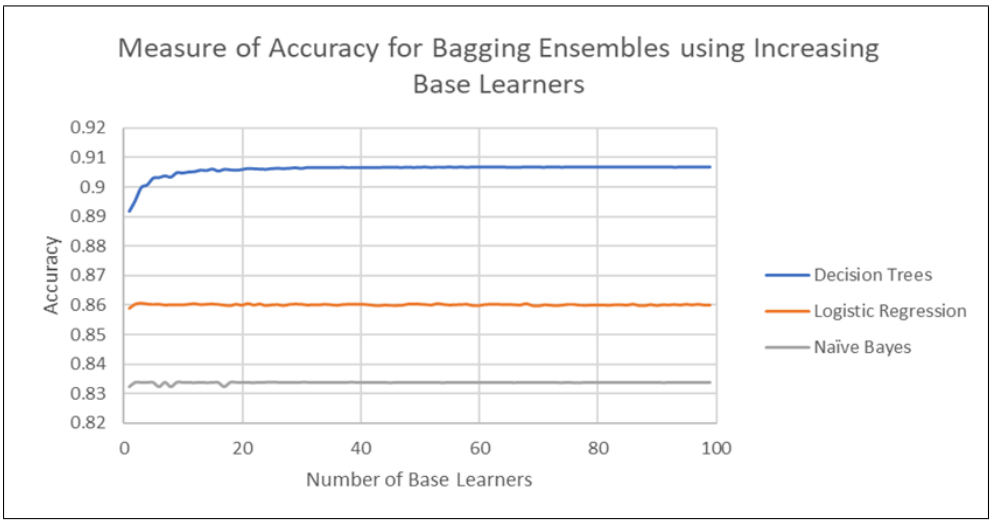

FIGURE 2. Bagging test results for increasing number of base learners

As seen in Figure 2, the increasing number of base learners does not significantly improve the accuracy of the ensemble model. However, there is a slight increase at the start of each curve. This upturn proves the theory that an ensemble learning would produce a higher accuracy compared to an individual model.

In terms of bagging, as shown in Table 4, it can be said that the decision trees make the best-bagged ensemble. Specifically having 61 trees within the ensemble produces the highest evaluation metrics compared to the two other base learner models. These ideal amounts for each bagged model were then implemented in the stacking methodology. The precision value is also the highest with the decision trees. This value is important as it indicates the model's ability to predict positive values. Compared to the other base learners, decision trees produce a significantly higher value. Therefore, a more viable prediction can be made with it. This point is further proven with the AUC score being the highest. This value indicates the model's ability to distinguish one class from the other. With a significantly higher AUC score, the decision tree ensemble has the best ability to classify whether an application is malicious or benign. Overall, through the bagging method, having a series of decision tree base learners would produce the most effective detection model.

Another approach in ensemble learning is the stacking methodology in which a prediction is made by a group of base learners, called the intermediate learners. After the intermediate predictions are made, an additional model is used on top to make the final decision, also noted here as the level 1 aggregator. The accuracy results are affected by having different level 1 aggregators, as can be seen in Table 5.

TABLE 5. Stacking parameter values with the highest accuracy

Best values from stacking					
Scenario	**# of Decision Trees**	**# of Logistic Regression Classifiers**	**# of Naïve Bayes Classifiers**	**Level 1 Aggregator**	**Accuracy**
1	1	1	1	Logistic Regression	0.8801
2	1	1	1	Naïve Bayes	0.8569
3	1	1	1	Decision Trees	0.8592
4	61	3	9	Logistic Regression	0.8839
5	61	3	9	Naïve Bayes	0.8569
6	61	3	9	Decision Trees	0.8561

As seen in Table 5, an ensemble with one base learner each was implemented with the three main models as the level one aggregator. Through the accuracies, it can be said that the Logistic Regression classifier produces the best accuracy with an 88.01% detection rate. This result can also be seen in the experimentation where the number of base learners was based on the best values obtained from bagging which then yielded an 88.39% detection rate.

With the best level one aggregator decided, an experimentation to see the effects of increasing the number of base learners in parallel was also conducted. The results to this experimentation can be seen in Figure 3 which shows the trend of the accuracies produced. It is seen that there is a sharp increase at the beginning of the experimentation, though as more base learners are added, it is shown to have no significant effect. It could also be said that there is no prominent trend that can be distinguished from increasing the number of each base learner. However, it is observed that once there are 94 of each base learner, the accuracy becomes the same and the trend flatlines.

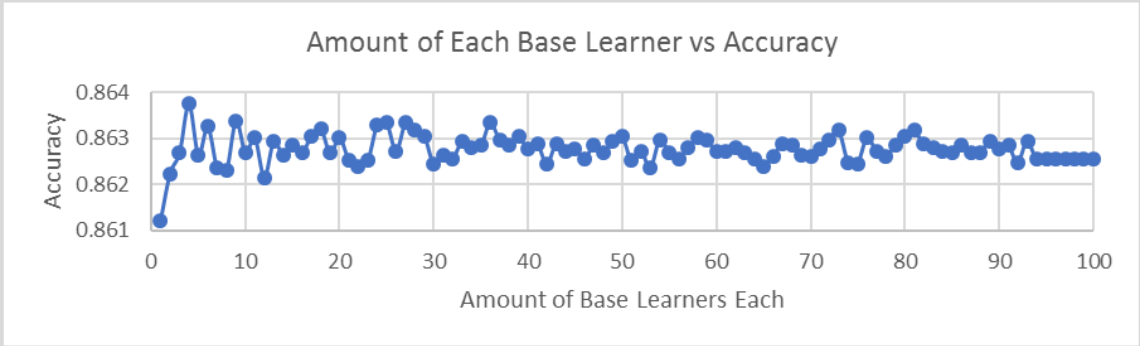

FIGURE 3. Test result for increasing the number of base learners

Based on this experimentation result shown in Table 6, it can be inferred that 4 is the most optimal value in terms of obtaining the highest accuracy when stacking the same amount of each base learner. Having 4 of each base learner and consequently stacked with a logistic regression classifier is seen to produce the highest accuracy with 86.37%. This value is considered significant compared to the next highest few accuracy values which all revolved around 86.33%.

TABLE 6. Results for the best value for stacking the same amount of base learners

Best values from stacking the same amount each				
# of Decision Trees	**# of Logistic Regression Classifiers**	**# of Naïve Bayes Classifiers**	**Level 1 Aggregator**	**Accuracy**
4	4	4	Logistic Regression	0.86374964
9	9	9	Logistic Regression	0.86338195
25	25	25	Logistic Regression	0.86334110
24	24	24	Logistic Regression	0.86330024
6	6	6	Logistic Regression	0.86325939

4. **Conclusions and Future Works.** In conclusion, ensemble learning methods can be implemented to increase the accuracy of individual models. However, continuously increasing the number of base learners in the bagging methodology does not improve the overall performance of the model. On the other hand, the experimentation in the stacking methodology shows that not all aggregators can improve the accuracy. Overall,

it is found that the best results gathered came from a homogeneous ensemble model which implemented the bagging ensemble technique that aggregated 61 decision trees. The individual learners within the model each used a max depth of 56 and the random state of 44. This best model was also proven to be successful based on its accuracy of 90.67%, AUC value of 84.44%, and precision of 91.10% in classifying malware from benign apps.

For the future work, the research can be extended to give a primary focus on minimizing the false negative rate to optimize recall. This can be beneficial as when it comes to detecting malware, the effect of a false negative becomes significantly more detrimental. Furthermore, through the implementation of external datasets and analysis, the detection model could also be improved to have the ability to determine the type of malware from the permissions an application requests.

REFERENCES

[1] A. Possemato, S. Aonzo, D. Balzarotti and Y. Fratantonio, Trust, but verify: A longitudinal analysis of Android OEM compliance and customization, *2021 IEEE Symposium on Security and Privacy (SP)*, pp.87-102, DOI: 10.1109/SP40001.2021.00074, 2021.

[2] A. Naway and Y. Li, A review on the use of deep learning in Android malware detection, *International Journal of Computer Science and Mobile Computing*, vol.7, no.12, pp.42-58, 2018.

[3] N. Nouman, Z. Noreen and F. Naz, Vulnerabilities in Android OS: Challenges and mitigation techniques, in *Digital Technologies and Applications*, S. Motahhir and B. Bossoufi (eds.), Cham, Springer International Publishing, DOI: 10.1007/978-3-031-01942-5_25, 2022.

[4] A. Fatima, R. Maurya, M. K. Dutta, R. Burget and J. Masek, Android malware detection using genetic algorithm based optimized feature selection and machine learning, *2019 42nd International Conference on Telecommunications and Signal Processing (TSP)*, pp.220-223, DOI: 10.1109/TSP.2019.8769039, 2019.

[5] N. S. Kumar, P. V. Vardhan, P. A. Chandra, S. V. S. Kumar and T. N. P. Madhuri, Analysis of malware in Android features using machine learning, *Journal of Engineering Sciences*, vol.13, pp.19-28, DOI: 10.15433.JES.2022.V13I01.43P.3, 2022.

[6] S. Sabhadiya, J. Barad and J. Gheewala, Android malware detection using deep learning, *2019 3rd International Conference on Trends in Electronics and Informatics (ICOEI)*, pp.1254-1260, DOI: 10.1109/ICOEI.2019.8862633, 2019.

[7] U. S. Jannat, S. M. Hasnayeen, M. K. B. Shuhan and M. S. Ferdous, Analysis and detection of malware in Android applications using machine learning, *2019 International Conference on Electrical, Computer and Communication Engineering (ECCE)*, pp.1-7, DOI: 10.1109/ECACE.2019.8679493, 2019.

[8] M. Cao, K. Ahmed and J. Rubin, Rotten apples spoil the bunch: An anatomy of Google Play malware, *2022 IEEE/ACM 44th International Conference on Software Engineering (ICSE)*, pp.1919-1931, DOI: 10.1145/3510003.3510161, 2022.

[9] V. Tyagi, R. Singh, A. Tripathi, V. Agarwal and S. Pandey, An Android APP permission analysis for user privacy and security, *Advances in Systems Analysis, Software Engineering, and High Performance Computing*, IGI Global, pp.89-103, DOI: 10.4018/978-1-6684-4225-8.ch006, 2022.

[10] A. Amin, A. Eldessouki, M. T. Magdy, N. Abdeen, H. Hindy and I. Hegazy, Androshield: Automated Android applications vulnerability detection, a hybrid static and dynamic analysis approach, *Information*, vol.10, no.10, 326, DOI: 10.3390/info10100326, 2019.

[11] T. Kim, B. Kang, M. Rho, S. Sezer and E. G. Im, A multimodal deep learning method for Android malware detection using various features, *IEEE Transactions on Information Forensics and Security*, vol.14, no.3, pp.773-788, DOI: 10.1109/TIFS.2018.2866319, 2019.

[12] R. B. Hadiprakoso, H. Kabetta and I. K. S. Buana, Hybrid-based malware analysis for effective and efficiency Android malware detection, *2020 International Conference on Informatics, Multimedia, Cyber and Information System (ICIMCIS)*, pp.8-12, DOI: 10.1109/ICIMCIS51567.2020.9354315, 2020.

[13] D. Arp, M. Spreitzenbarth, M. Hübner, H. Gascon and K. Rieck, Drebin: Effective and explainable detection of Android malware in your pocket, *Network and Distributed System Security Symposium (NDSS)*, pp.720-731, DOI: 10.14722/ndss.2014.23247, 2014.

[14] D. J. Wu, C. H. Mao, T. E. Wei, H. M. Lee and K. P. Wu, Droidmat: Android malware detection through manifest and API calls tracing, *2012 7th Asia Joint Conference on Information Security*, pp.62-69, DOI: 10.1109/AsiaJCIS.2012.18, 2012.

[15] J. T. McDonald, N. Herron, W. B. Glisson and R. K. Benton, Machine learning-based Android malware detection using manifest permissions, *Proc. of the Annual Hawaii International Conference on System Sciences*, pp.6976-6985, DOI: 10.24251/HICSS.2021.839, 2021.

[16] A. Kapratwar, F. D. Troia and M. Stamp, Static and dynamic analysis of Android malware, *Proc. of the 3rd International Conference on Information Systems Security and Privacy (ICISSP2017)*, pp.653-662, DOI: 10.5220/0006256706530662, 2017.

[17] M. Ashawa and S. Morris, Analysis of Android malware detection techniques: A systematic review, *International Journal of Cyber-Security and Digital Forensics*, vol.8, pp.177-187, DOI: 10.17781/P002605, 2019.

[18] K. Liu, S. Xu, G. Xu, M. Zhang, D. Sun and H. Liu, A review of Android malware detection approaches based on machine learning, *IEEE Access*, vol.8, pp.124579-124607, DOI: 10.1109/ACCESS.2020.3006143, 2020.

[19] H. J. Zhu, T. H. Jiang, B. Ma, Z. H. You, W. L. Shi and L. Cheng, HEMD: A highly efficient random forest-based malware detection framework for Android, *Neural Computing and Applications*, vol.30, pp.3353-3361, DOI: 10.1007/s00521-017-2914-y, 2017.

[20] H. Sayadi, N. Patel, P. D. S. Manoj, A. Sasan, S. Rafatirad and H. Homayoun, Ensemble learning for effective run-time hardware-based malware detection: A comprehensive analysis and classification, *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*, pp.1-6, DOI: 10.1109/DAC.2018.8465828, 2018.

[21] P. Illy, G. Kaddoum, C. M. Moreira, K. Kaur and S. Garg, Securing fog-to-things environment using intrusion detection system based on ensemble learning, *2019 IEEE Wireless Communications and Networking Conference (WCNC)*, pp.1-7, DOI: 10.1109/WCNC.2019.8885534, 2019.

[22] S. Y. Yerima, S. Sezer and I. Muttik, High accuracy Android malware detection using ensemble learning, *IET Information Security*, vol.9, pp.313-320, DOI: 10.1049/iet-ifs.2014.0099, 2015.

[23] K. Shao, Q. Xiong and Z. Cai, FB2droid: A novel malware family-based bagging algorithm for Android malware detection, *Security and Communication Networks*, pp.1-13, DOI: 10.1155/2021/6642252, 2021.

[24] H. Zhu, Y. Li, R. Li, J. Li, Z. You and H. Song, SEDMDroid: An enhanced stacking ensemble framework for Android malware detection, *IEEE Transactions on Network Science and Engineering*, vol.8, no.2, pp.984-994, DOI: 10.1109/TNSE.2020.2996379, 2021.

[25] A. Mahindru, Android permissions dataset, Android malware and benign application data set (consist of permissions and API calls), *Mendeley Data*, vol.3, DOI: 10.17632/b4mxg7ydb7.3, 2020.