JOIV : Int. J. Inform. Visualization, 6(1) - March 2022 138-144

INTERNATIONAL JOURNAL ON INFORMATICS VISUALIZATION

journal homepage : www.joiv.org/index.php/joiv

An Android Malware Detection System using a Knowledge-based Permission Counting Method

Sun-A Lee [a], A-Reum Yoon [a], Ji-Won Lee [a], Kwangjae Lee [a,*]

[a] *Department of Information Security Engineering 31, Sangmyeongdae-gil, Dongnam-gu, Cheonan-si, Chungcheongnam-do, 31066, Republic of Korea*

Corresponding author: [*]begleam@smu.ac.kr

***Abstract*— As the number of damage cases caused by malicious apps increases, accurate detection is required through various detection conditions, not just detection using simple techniques. This paper proposes a knowledge-based machine learning method using authority information and adding its usage counting features. This method classifies training apps and malicious apps through machine learning using permission features in manifest.xml of Android apps. As a result of the experiment, accuracy, recall, precision, F1 score are 99.01%, 97.70%, 100.0%, 99.01%, respectively. Since recall is higher than other indicators, it accurately predicts malicious apps as malicious. In other words, the proposed system effectively prevents the distribution of malicious apps. As the number of harmful apps develops daily, it was determined in this study that it is critical to detect malicious apps using a machine learning model effectively. However, utilizing permission alone as a criterion for distinguishing between legitimate and malicious apps is insufficient to detect all harmful apps that emerge from new attack technologies. Combining feature information efficient in detecting malicious apps, such as APIs that access and control sensitive data from users or adding other detection criteria will likely improve the detection model's accuracy. According to the upcoming study, recent attackers have used obfuscation to disguise harmful code and hinder static analysis of rogue programs. It is important to consider how to detect harmful apps that are obfuscated in this way.**

***Keywords*—Machine learning; android malware detection; permission counting; knowledge-based analysis.**

Manuscript received 22 Oct. 2021; revised 17 Nov. 2021; accepted 21 Dec. 2021. Date of publication 31 Mar. 2022.

I. INTRODUCTION

As the use of smartphones has increased recently, damage from malicious apps is also increasing. Malicious apps refer to malicious software (Malware) that performs malicious functions by disguising it as a normal app on a smartphone [1], [2]. According to the Korea Internet & Security Agency (KISA), malicious apps increased 5.5 times from 1,635 in 2016 to 9,051 in 2019 and are being used for intelligent crimes [3]. Additionally, according to the EST-Security report, which analyzed the sources of apps installed on over 12 million Android devices, 67% of malicious apps were installed on the Google Play Store [4]. Despite the tricky upload key acquisition and signature procedures when uploading apps to the Google Play Store, it has not been able to escape from the wrong explanation that it is a malicious app installation platform. The installed malicious apps are cleverly distributed to smartphone users, causing financial losses. For example, there were financial crimes such as stealing personal information from smartphones or transferring money from accounts to defraud 264 million won [5]. The representative research to prevent damage caused by malicious apps is a signature analysis method. This analysis consists of a code-based static analysis and a sandbox-based dynamic analysis [6]–[8]. The static analysis can detect the same type of malicious apps by analyzing the source code of malicious apps. However, it is difficult to detect manual analysis time problems and new types of malicious apps because their signatures change. The dynamic analysis analyzes suspected malicious output actions and packets to determine whether it is malicious. Thus, it can detect malicious apps that are difficult to perform static analysis, such as obfuscation. However, it is possible to avoid detection with the execution environment detection or the conditional operation function of malicious apps. In that case, it may be difficult to detect malicious apps. In addition, like static analysis, it is difficult to detect new types of malicious apps.

In recent research, machine-learning (ML) based detection methods have been proposed to solve new mutation detection and false detection rates. These methods distinguish between a normal app and a malicious app using a feature that is a form

or change of an input value. If we reprocess information from thousands or millions of malicious apps in our dataset and apply it to training to create a model, it becomes a ML-based model with specific features and has a high detection rate [9], [10]. The ML-based researches performed detection by reading the information in the Android Application Package (APK) file and using permission feature or malicious behavior mainly used in malicious apps [11]–[15]. These research use the dataset, including the permission features of the android system. And to obtain high accuracy, their dataset adds features: API calls, Dex header, Broadcast Receiver, Service, etc. In addition, a study was conducted to increase the weight of the top 20 highly important inputs by organizing feature information on authority, API, etc., which are frequently used in malicious app detection. However, if there is a change in the input value due to obfuscation, the probability of incorrect detection increases significantly because it is difficult to determine with a trained model [8, 16]. In addition, the use of permission information, which is an input value that is not obfuscated, can reduce the false detection rate, but this single feature extraction method has a limitation in that the detection rate is low.

In this paper, we propose a method of lowering the false detection rate as a machine learning method that extracts and uses permission information that is resistant to obfuscation. In more detail, for improving the low detection rate of single feature extraction, the top 20 permissions analyzed as important in detecting malicious and normal apps, and the number of permissions used are additionally created through frequency analysis. The proposed method is not only tough on false detection by obfuscation but also has a higher detection rate than the existing single feature extraction method.

A. APK Configuration

APK is a package file used to distribute Android software and middleware. This file contains elements that are needed to run the app, such as AndroidManifest.xml, Class. dex, Res, and Lib. The description of the components is shown in Table 1 [17].

TABLE I
COMPONENTS OF AN APK FILE

Name	Description
Android-Manifest.xml	Xml file that manages the app. Application Permission, Intent, Service, Activity, SDK version information.
Classes.dex	Collected class-files and converted them into byte code to allow Android Dalvik virtual machines to recognize elements.
resources.arsc	A file containing resource file information. Save type and id information for various resource files.
/res	Uncompiled images. A folder containing xml resource files.
/lib	A folder containing the library. Composed of .so files compiled appropriately for each process created with NDK (Native Development Kit).
/assets	A folder containing app information that can be managed by Assets-Manager.
/META-INF	A folder related to the signature. Save SHA-1 and Base-64 signature values.

This paper extracts the app's permission information using Android static analysis. Among the Android app components, the AndroidManifest file has the permission information needed to operate the app. It is used to protect the user's personal information, and a system is automatically assigned according to the authority, and user approval is required [18]. In addition, malicious apps excessively require authorization information [12]. Therefore, a series of permission information that may be used in malicious behavior, such as accessing important information on a smartphone or exchanging data over the Internet, may be used as a malicious app detection feature.

B. Malicious App Dataset

This paper uses the Android Malicious App Dataset (CIC-AndMal2017) provided by the Canadian Institute for Cybersecurity (CIC) [19]. The dataset is a collection of 429 malicious apps and 5,065 normal apps from the Google Play Store on smartphones to collect traffic generated through various scenarios such as Internet searches, phone calls, and messages. Based on that information, it was classified into 42 malicious software groups and four categories (Adware, Ransomware, Scareware, and SMS-malware). Categories of datasets and malicious software groups can be represented as shown in Table 2 [20], [21], [22], [23].

TABLE II
CATEGORIES AND MALWARE FAMILIES IN ANDMAL2017

Name	Description Malware Group (Family)
Adware	Software that arbitrarily displays advertising to users.
	Dowgin, Ewind, Feiwo, Gooligan, Kemoge, koodous, Mobidash, Selfmite, Shuanet, Youmi family
Ransomware	Malware that infiltrates under the guise of e-mail or updates, encrypts data on the user's device, and requires payment in return for decryption.
	Charger, Jisut, Koler, LockerPin, Simplocker, Pletor, PornDroid, RansomBO, Svpeng, annaLocker family
Scareware	A copycat version of ransomware. Stressing that they have control of the computer and demand money.
	AndroidDefender, AndroidSpy.277, AV for Android, AVpass, FakeApp, FakeApp.AL, FakeAV, FakeJobOffer, FakeTaoBao, Penetho, VirusShield family
SMS-malware	SMS+ Phishing. It induces malicious app installation by impersonating normal apps that are generally installed on mobile phones.
	BeanBot, Biige, FakeInst, FakeMart, FakeNotify, Jifake, Mazarbot, Nandrobox, Plankton, SMSsniffer, Zsone family

C. Machine-learning Algorithm

Machine learning algorithms are largely divided into a supervised learning method of learning computers with correct labels on training data and an unsupervised learning method of learning computers without correct labels on training data. Supervised learning is a method of classifying new data by learning within a predetermined label using classification or regression, and unsupervised learning is used

to obtain meaningful knowledge through data, although there is no prior knowledge of specific results such as clustering or pattern recognition [24], [25].

This paper uses a ML algorithm for classifying normal and malicious apps, using a supervised learning method. To select the right algorithm for the proposed system, Representative classification algorithms such as K-Nearest Neighbor (K-NN), Support Vector Machine (SVM), Ada Boost, Extra Tree, and Random Forest were compared and analyzed in Table 3 [26]–[30].

TABLE III
MACHINE LEARNING CLASSIFICATION USING THE SUPERVISED LEARNING

Algorithm	Description
K-NN	A method of classifying new input data into the proximity of the neighboring data category.
SVM	Define the classification baseline, the decision boundary, as a model. A method of categorizing which side of a boundary the new data belongs to.
Ada-Boost	As a type of ensemble learning, a method of classifying weak classifiers into strong classifiers by combining the results. The weight of the sample misclassified by the drug classifier is applied according to the situation.
Extra-Tree	Ensemble learning method that randomly generates N Weak Trees for existing datasets and selects classifiers with good performance by combining classification results.
Random-Forest	Create N Weak Trees randomly while allowing duplication for the dataset. Ensemble learning method of selecting a classifier with good performance by combining classification results.

D. *Performance Evaluation Index*

In this paper, a confusion matrix and a receiver operating characteristic (ROC) curve are used as performance evaluation indicators for machine learning models [31], [32]. The confusion matrix consists of True Positive (TP), False Positive (FP), True Negative (TN), and False Negative (FN). In order to minimize misdetection and misdetection, the smaller the number of FPs and FNs, the better the classification model. In addition, four evaluation indicators are added: Accuracy, Precision, Recall, and F1-score through TP, FN, FP, and TN of the confusion matrix shown in Table 4 [33].

TABLE IV
CONFUSION MATRIX

		Predicted	
		Malicious App	**Benign App**
Actual	**Malicious App**	True Positive	False Negative
	Benign App	False Positive	True Negative

Accuracy represents the ratio of the number of normal detections in the total detected data. Precision is the percentage of the actual number of malicious apps predicted by malicious apps. On the other hand, Recall is the ratio of the number predicted by malicious apps among the actual malicious apps. Since Precision and Recall have a relationship that is difficult to balance at the same time, F1-score, which is a harmonious average of precision and recall, is required. In addition, an ROC curve and an Area under the ROC curve (AUC) were added as performance evaluation indicators, which show the performance of the classification model as a curve and indicate the area of the curve. The AUC-ROC curve represents the relationship between sensitivity and specificity on a two-dimensional plane. The criteria for how well you find malicious apps are expressed as sensitivity (Y-axis), and the criteria for how well you classify normal apps are expressed as specificity (X-axis) [34]. This curve represents a model with higher classification accuracy as it approaches the upper left (0, 1) of the coordinates. Finally, a K-Fold Cross Validation technique was applied to verify the reliability of the detection performance evaluation. For K-layer Cross-Validation, as shown in Fig. 1, after arbitrarily dividing the dataset into the same size, one of them is used as a validation dataset and the other (K-1) as a learning dataset. If this process is repeated k times sequentially, it is possible to verify the entire given dataset [35].

In this paper, for the performance evaluation of the proposed model, the k value was set to 8, divided into 8 datasets and performance evaluation was performed.

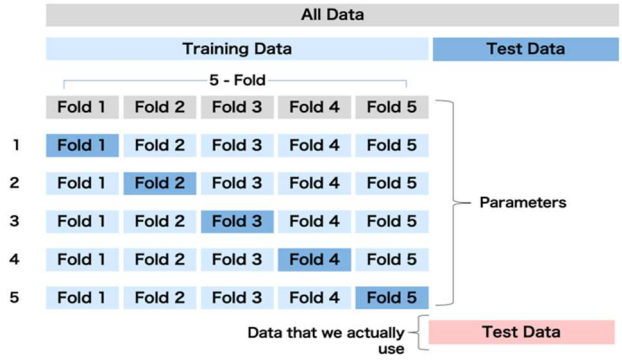

Fig. 1 An example of the K-fold Cross Validation.

II. MATERIAL AND METHOD

Fig. 2 shows the conceptual diagram of the Android malicious app detection system proposed in this paper. Various files exist inside packages of normal and malicious apps. The APK file consists of several files required to run the app, of which the AndroidManifest.xml file has the necessary permission information to run the app.

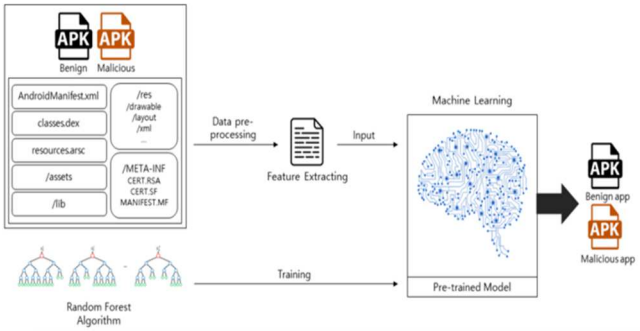

Fig. 2 A Conceptual Diagram of a Proposed Android Malware Detection System.

In Android systems, permissions used in apps must be licensed to protect users' personal information. Depending on the type of permission, the user directly approves it, or the system is automatically granted. A typical feature of malicious apps is that they excessively require the use of permissions. Therefore, we propose a ML-based detection method that classifies normal and malicious apps using permission information. First, the privileged information of the app is extracted in a one-hot encoding format from the AndroidManifest.xml file. Input data for training the ML model is converted into a dataset in the form of a csv file. In order to increase detection performance, the frequency of data is analyzed, and the total number of permissions (TPC) and the number of frequently used permissions Main Permission Count (MPC) are learning algorithm is designed by comparing binary classification algorithms and selecting the one with good performance. Finally, a new app that requires a malicious or normal test is determined using the ML model previously created.

A. Dataset Creation

The preprocessing process must inevitably perform repetitive tasks. This is because it is the process of creating a dataset necessary for training with numerous related materials. Therefore, in this paper, an automation program described in Algorithm 1 was used to extract permission information from multiple apps automatically. First, an existing permission list file is read, and a label variable is created. Additionally, create the value variable to have the label size +3 as a column. When the column size of the generated value is n, 1 is classified into an order, 2 is an APK name, 3 to n-1 is a permission list, and n is classified according to the index. Next, it is repeated as many times as the number of APKs prepared to create a dataset. A new line of value is added, and the class value is stored as 1 malicious and 0 normal depending on whether the APK file is malicious. Then add the value item to 1 for all the permissions in the AndroidManifest.xml file. In this case, if an authority that is not in the permission list is found, a new column is added to column n-1 of value and set to 1. The label also inserts the permission name into the n-1 index as a value. After that, when the repetition ends by the number of APK files, a dataset is generated by combining label and value and stored as a CSV (Comma-Separated Values) file. Finally, save the updated label in this task to the permission list file.

Algorithm 1 Auto Extract Algorithm

```
Input: apk, APK files
Input: permList, permission list file
Output: dataset, dataset file in csv format
Output: permList, permission list file
Method:
label ← permList
n ← #permList + 3
for j=1 to #apk do
  if value is not exist then
    create value to 1×n array
  else then
    attach new row of value
  decompression apk[j]
  value[j][1] ← j
  value[j][2] ← name of apk
  for k=3 to n-1 do
    if perm does not exist in label then
      insert new column at n-1 index of value
      insert perm's name at n-1 index of label
      n ← n + 1
    if label[k-2] exist among all perms in apk[k] then
      value[j][k] ← 1
  end for
  value[j][n] ← (apk == malicious app) ? 1 : 0
end for
dataset ← (label, value)
permList ← label
```

B. Adding Frequency Features to Datasets

Since the single feature extraction method has a low detection rate by detecting only authority information, the performance is further improved by using the frequency analysis results. The two features define features that mean the total number of permissions used in the app as TPC, and features that mean the top 20 permissions analyzed as important in using the app as MPC. Adding these features is also iterative, so an automation program described in Algorithm 2 was used.

TABLE V
TOP 20 PERMISSIONS

No	Permission (Description)
1	android.permission.READ_PHONE_STATE (Read about phone status such as device phone number, network information, and call status in progress)
2	android.permission.INSTALL_SHORTCUT (Install icons on the home screen)
3	android.permission.SYSTEM_ALERT_WINDOW (Open windows using top TYPE_SYSTEM_ALERT of other applications)
4	android.permission.GET_TASKS (Access current or recently executed task information)
5	android.permission.ACCESS_WIFI_STATE (Access to information about Wi-Fi networks)
6	android.permission.MOUNT_UNMOUNT_FILESYSTEMS (File system format for removable storage)
7	com.google.android.c2dm.permission.RECEIVE (Receive messages from c2dm server)
8	android.permission.WRITE_EXTERNAL_STORAGE (Write a file to an external repository)
9	android.permission.ACCESS_COARSE_LOCATION (Access to a wide range of locations (Cell-ID, WiFi))
10	android.permission.CHANGE_WIFI_STATE (Change Wi-Fi connection status)
11	android.permission.VIBRATE (Vibration control)
12	com.android.vending.BILLING (Access to payment data)
13	android.permission.ACCESS_FINE_LOCATION (Access to GPS)
14	android.permission.WAKE_LOCK (Keep the process when the screen is dark or on standby)
15	android.permission.READ_EXTERNAL_STORAGE (Read a file to an external repository)
16	android.permission.RECEIVE_BOOT_COMPLETED (Boot complete execution)
17	android.permission.GET_ACCOUNTS (Access the account list from within the account service)
18	android.permission.CAMERA (Access to camera equipment)
19	android.permission.ACCESS_NETWORK_STATE (Access to information about network access)
20	com.google.android.gsf.permission.READ_GSERVICES (Read data about map)

First, the dataset file is read and stored in label and value variables, respectively. And the top 20 authority names of high importance are defined as permList. Insert two additional columns in column n-1 of value and insert TPC and MPC into the n-1 index as values in the label. In order to, obtain the

frequency, TPC and MPC variables are created and initialized to zero. Next, since one line of the dataset extracts features of one APK file, the number of lines of value repeats it. TPC checks all the characteristics of each APK and accumulates the permission's true (1)/false (0) values.

Algorithm 2 Auto Add Feature Algorithm

```
Input: dataset, dataset file in csv format
Input: permList, top 20 permission list to detect malware
Output: dataset, dataset file in csv format
Method:
(label, value) ← dataset
insert new 2-column at n-1 index of value
insert (tpc's name, mpc's name) at n-1 index of label
tpc ← 0, mpc ← 0
for j=1 to #value's row do
   for k=3 to n-3 do
      tcp ← tcp + value[j][k]
      if label[k] exist among all permList then
         mpc ← mpc + value[j][k]
   end for
   value[j][n-2] ← tpc, value[j][n-1] ← mpc
   tpc ← 0, mpc ← 0
end for
dataset ← (label, value)
```

The MCP verifies all the privileged names of each APK. After checking whether the value is in the permList, the authority's true (1)/false (0) values are accumulated and summed, if applicable. Then, TPC and MPC values are stored in the n-2 and n-1 indexes of the value and are initialized to zero again. After that, if the number of lines of value is repeated, combine the label and value to create a dataset and save it as a CSV file. A total of 1013 APK files and 1031 Permission Lists were sorted. If the corresponding permission exists in each APK file, it is output as 1 and if not, it is output as 0. In addition, it is possible to check the frequency of the total permission present in each APK file in the last cell. The top 20 Permission frequencies based on the importance of feature information were added to the existing csv file. If there are the top 20 permissions based on the importance of feature information among the permissions present in each APK file, it is vectorized to check the frequency by accumulating them

C. Creation of Classification Algorithm Models

Six hundred downloaded datasets were collected in Benign folders, 413 malicious apps in Malicious folders, and 1031 Permission features used in normal and malicious apps were extracted. The Permission frequency contained in one APK file was converted into a csv file. For higher accuracy of normal and malicious apps classification, the top 20 Permission frequencies based on the importance of feature information in one APK file were converted to csv files. A classification algorithm selection process was performed based on the previously extracted authority information to determine normal and malicious apps. K-NN, SVM, Ada Boost, Extra Tree, and Random Forest were considered as classification algorithms as shown in Table VI.

TABLE VI
MACHINE LEARNING MODELS ACCURACY COMPARE

Model	Accuracy	Precision	Recall	F1 score
K-NN*1	90.79	87.18	88.70	90.79
SVM*2	90.79	90.65	84.35	90.79
AdaBoost*3	93.75	92.86	90.43	93.75
Extra Tree*4	94.74	93.04	93.04	94.74
RF*5	95.07	93.10	93.91	95.07

*1 K-NN options: n_neighbors =10
*2 SVM options: C= 0.1
*3 Ada-Boost options: n_estimators=100
*4 Extra-Tree: n_estimators=100
*5 Random Forest options: n_estimators= 50

As a result, Random Forest was selected as an algorithm suitable for this study. Random Forest can prevent overfitting by the law of large numbers made of randomness. It is also robust to noise and reduces predicted volatility. The binary classification ML model was trained through the previously generated dataset, and the performance of the five classification algorithms was compared. Hyper-Parameter of each classification model showed optimal performance even with default values, so only the n_estimators' values of three models, Random Forest, Extra Tree, and Ada Boost, were set differently. In the case of K-NN, the n_neighbors value was set to the optimal 10 in the experiment, and the SVM gave the gamma C value to 0.1. Ada Boost set it to 100 optimal for the experiment, and Random Forest and Extra Tree set it to 50 optimal for the experiment. As a result, comparing the accuracy with the training rate of 80% and the verification rate of 20% for the experimental data confirmed that the Random Forest model showed the highest performance with 95.07% accuracy in determining normal and malicious apps.

III. RESULT AND DISCUSSION

In this experiment, 600 normal apps and 413 malicious apps were used. Among the permission information extracted from a total of 1031 APK files, a total of 1013 feature data were used for model training by deleting duplicated or meaningless data. The division ratio of the training and verification datasets was 8:2, and the training datasets were used as 810 and 203 testing datasets. The experiment was conducted by adding frequency to the previously created dataset using the Random Forest algorithm.

A. Frequency Feature for Permission (EXP #1)

This study considers whether it is possible to distinguish between normal and malicious apps with higher accuracy by adding features on the frequency of permissions in APK files. This experiment identifies the frequency of permission in one APK file and uses the information as a feature of the ML model to increase classification accuracy. The experiment was conducted with a total of 1032 features by adding a field for the frequency of the permission of the corresponding APK to the pre-treated experimental data.

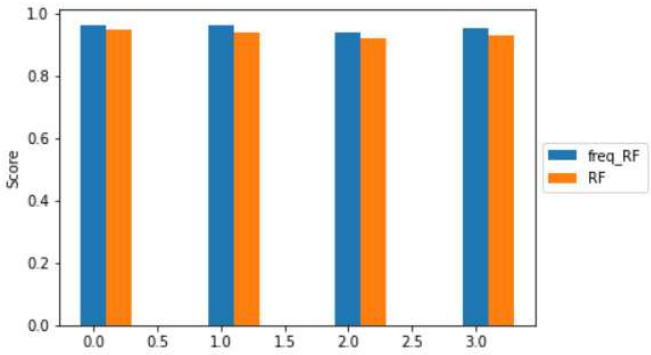

Fig. 3 Compare of Permission Frequency Performance Indicator

Earlier, as it was confirmed that the performance of the Random Forest model was the best through the algorithm selection process, the experiment was conducted using only the Random Forest model. The experimental data was divided by a verification ratio of 20%, and the value of Hyper-Parameter optimal for the experiment was set. The max_depth value was set to 50, the min_samples_split value was set to 5, and the n_estimators value was set to 50.

The algorithm selection process confirmed that the result of the normal/malicious app determination of the ML model training without adding the Permission frequency was 95.07%. As a result of training the machine learning model by adding features on the frequency of permission of each APK file, the accuracy was 97.04%. Compared with previous results that did not add the frequency of permission as a feature, it can be seen that the accuracy was improved by 1.97%.

A total of four types of model performance evaluation indicators were used: accuracy, accuracy, reproduction rate, and F1-score. In EXP #1, the accuracy of the ML model was 97.04%, the precision was 97.65%, and the reproduction rate was 95.40%. The F1-score, which calculated the harmonic mean of precision and reproduction rate, also showed a result of 97.04%. Earlier, as a result of calculating the accuracy with 8-layer cross-validation for reliable evaluation, the verification result was 93.48%.

B. frequency feature for the top 20 permissions (EXP #2)

Through EXP #1, it was confirmed that the accuracy of determining normal and malicious apps according to the frequency of permission was improved. In this experiment, the frequency of permissions frequently used in apps is added as a feature to increase the classification accuracy of normal and malicious apps. Even if many permissions make up malicious apps, there may be cases where normal apps request permission. In order to prevent discriminating normal apps as malicious apps, this study improves the accuracy of determining normal apps and malicious apps by adding frequency features, not the presence or absence of the top 20 permissions.

In EXP #2, an experiment was conducted with a total of 1033 features by adding the frequency of use of the top 20 Permissions shown in the above feature information important to the dataset used in EXP #1. The experimental data were divided by a verification ratio of 20%, and the Hyper-Parameter value was set to a max_depth value of 50, a value of min_samples_split, and a value of n_estimulators of 50, which are optimal for the experiment.

The training was conducted by setting the same algorithm and the same parameter, and the accuracy was 99.01%, which showed higher classification accuracy than EXP #1. The reproduction rate was 97.70%, but the reproduction rate was 100%. Since the False Positive ratio is 0, there is no false detection. F1-score also showed high results at 99.01%. To compare under the same conditions as EXP #1, the EXP #2 model was classified with an accuracy of 94.18% due to 8-layer cross-validation.

C. Comparison Between Experiments

As a result of evaluating the two experiments with the same performance evaluation index, the Performance of EXP #2 is relatively superior to that of EXP #1. Table 7 is the result of synthesizing the previous two experimental results. In terms of accuracy and F1-score, EXP #2 showed 1.97% higher accuracy than EXP #1, and there was a 0.05% fine difference in terms of precision. However, the reproduction rate was 4.6%, showing the biggest difference. This is because there was a difference between the two models in the False Positive ratio in the confusion matrix.

In Fig. 4, the two experimental results are shown and compared as a ROC curve. Although there is a slight difference, the ROC curve corner of EXP #2 is closer to the upper left. The AUC value, which means the area under the ROC curve, can also be determined by classification accuracy, with the AUC value of the EXP #1 model being 0.9975 and the EXP #2 model being 0.9978. The larger the area under the ROC curve, the better the model, so the EXP #2 model with a relatively high AUC value is the model optimized for classification. In addition, the EXP #2 model was classified with 0.7% higher accuracy in the 8-layer cross-validation conducted to increase the reliability of the experimental results further.

TABLE VII
EXP 1, EXP 2 COMPARISON OF RESULTS

Model	Accuracy	Precision	Recall	F1 score
Exp #1*	97.04	97.65	95.4	97.04
Exp #2*	99.01	97.70	100.0	99.01

*Options: n_neighbors =100, max_depth=50, min_samples_splits =5

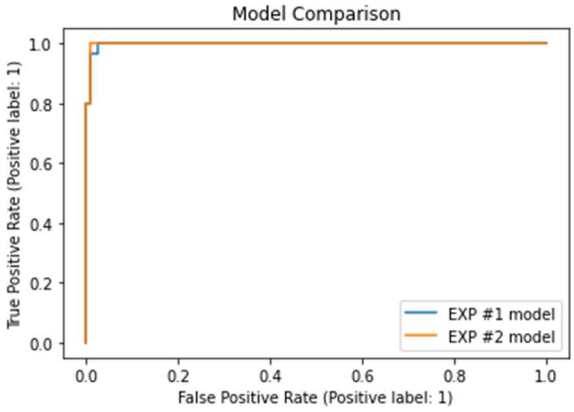

Fig. 4 The ROC Curves of EXP #1 and EXP 2.

IV. CONCLUSION

This study classified normal and malicious apps through ML-based detection techniques based on the frequency of permission of Android apps. Considering that malicious apps require excessive permission, unlike normal apps, we checked whether malicious apps could be detected more accurately by using them as frequency feature data using permission. Among the various ML classification algorithms, Random-forest showed the highest accuracy. The simple Permission-based detection model using Random Forest showed an accuracy of 95.07%, and the detection model, including the frequency at which the app used permission, obtained an accuracy of 97.04%. The trained model showed 99.01% accuracy, including the frequency of using the top 20 Permission information that affects distinguishing between

normal and malicious apps. It was confirmed that the accuracy was improved by about 2% by including meaningful feature data such as the frequency of the top 20 permissions based on the importance of feature information using the frequency of use of the permission. Since it is important to minimize misdetection and misdetection and make accurate judgments in detecting normal and malicious apps, an ML model optimized for classification has been implemented. In this study, as the number of malicious apps that develop day-by-day increases, it was judged that it is important to detect malicious apps through the ML model accurately. However, simply using permission as a criterion for distinguishing between normal and malicious apps is not enough to detect all malicious apps appearing with new attack technology. In addition, it is expected that higher accuracy of the detection model can be expected by combining feature information effective in detecting malicious apps, such as APIs that access and control sensitive data from users or adding other detection criteria. In future research, recent attackers bypass malicious app detection using obfuscation that hides malicious code and prevents static analysis of malicious apps. It is necessary to think about how to detect malicious apps with such obfuscation.

REFERENCES

[1] J. A. Odey, B. Ola, and I. Agbonlahor, "The Cyber Crime of Juice Jacking in Developing Economies: Susceptibilities, Consequences and Control Measures," *European Journal of Information Technologies and Computer Science*, vol. 1, no. 5, pp. 1-5, 2021.

[2] R. Vinayakumar, M. Alazab, K. P. Soman, P. Poornachandran, and S. Venkatraman, "Robust intelligent malware detection using deep learning," *IEEE Access*, vol. 7, pp. 46717-46738, 2019.

[3] Joon-ho Han, "Mobile Hacking Malicious Apps Surge," Oct. 7, 2020. [Online]. Available: https://www.hanjunho.com/35/?q=YToxOntzOjEyOiJrZXl3b3JkX3R5cGUiO3M6MzoiYWxsIjt9&bmode=view&idx=5050914&t=board.

[4] EST security, "Play Store identified as main distribution vector for most Android malware," Nov. 12, 2020. [Online]. Available: https://blo g.alyac.co.kr/3370.

[5] Y. J. Lee, "Voice phishing gang arrested for extorting $264 million by inducing malicious app installation," May. 31, 2021. [Online]. Available: https://www.news1.kr/articles/?4323269.

[6] S. E. Kang, H. S. Yoon, and S. H. Jung, "Design and Implementation of API Extraction Method for Android Malicious Code Analysis Using Xposed," *Journal of the Korea Institute of Information Security & Cryptology*, vol. 29, no. 1, pp. 105-115, Feb. 2019.

[7] G. Y. Kim, S. R. Kim, Y. J. Jeon, and J. S. Kim, "A Trend of Machine Learning for Android Malware Detection and Permission Based Android Malware Detection using Deep Learning," *Korean Society of Digital Forensics*, vol. 14, no. 3, pp. 316-326, Sep. 2020.

[8] A. Afianian, S. Niksefat, B. Sadeghiyan, and D. Baptiste, "Malware dynamic analysis evasion techniques: A survey," *ACM Computing Surveys (CSUR)*, vol. 52, no. 6, pp. 1-28, 2019.

[9] Z. Fang, J. Wang, J. Geng, and X. Kan, "Feature selection for malware detection based on reinforcement learning," *IEEE Access*, vol. 7, pp. 176177-176187, 2019.

[10] J. G. Joo, I. S. Jeong, and S. H. Kang, "An Optimal Feature Selection Method to Detect Malwares in Real Time Using Machine Learning," *Journal of Korea, Multimedia Society*, vol. 22, no. 2, pp. 203-209, Feb. 2019.

[11] Y. Chang, B. Liu, L. Cong, H. Deng, J. Li, and Y. Chen, "Vulnerability Parser: A Static Vulnerability Analysis System for Android Applications," *Journal of Physics: Conference Series*, vol. 1288, no. 1, Aug. 2019.

[12] H. W. Lee and H. S. Lee, "Optimal Machine Learning Model for Detecting Normal and Malicious Android Apps," *The Journal of the Internet of Things in Korea*, vol. 6, no. 2, pp. 1-10, Jun. 2020.

[13] M. J. Kim and J. C. Ryou, "Development of LLDB module for potential vulnerability analysis in iOS Application," *Journal of Internet Computing and Services*, vol. 20, no. 4, pp. 13-19, 2019.

[14] K. W. Lee, S. T. Oh, and Y. Yoon, "Modeling and Selecting Optimal Features for Machine Learning Based Detections of Android Malwares," *Thesis Collection of the Korean Society for Information Processing*, vol. 8, no. 11, pp. 427-432, Nov. 2019.

[15] V. Sihag, M. Vardhan, P. Singh, G. Choudhary, and S. Son, "De-lady: Deep learning based android malware detection using dynamic features," *Journal of Internet Services and Information Security (JISIS)*, vol. 11, no. 2, pp. 34-45, 2021.

[16] H. W. Lee and H. S. Lee, "Optimal Machine Learning Model for Detecting Normal and Malicious Android Apps," *Journal of The Korea Internet of Things Society*, vol. 6, no. 2, pp. 1-10, 2020.

[17] J. Park, T. Kim, Y. Shin, J. Kim, and E. Choi, "Design and Implementation of a Pre-processing Method for Image-based Deep Learning of Malware," *Journal of Korea Multimedia Society*, vol. 23, no. 5, pp. 650-657, 2020.

[18] Android Developer, "Manifest.Permission," [Online]. Available: https://developer.android.com/reference/android/Manifest.permission

[19] S. I. Imtiaz, S. ur Rehman, A. R. Javed, Z. Jalil, X. Liu, and W. S. Alnumay, "DeepAMD: Detection and identification of Android malware using high-efficient Deep Artificial Neural Network," *Future Generation computer systems*, vol. 115, pp. 844-856, 2021.

[20] B. H. Kim and M. T. Kwon, "Measures for Adware and Spyware," *Journal of Convergence Security*, vo1. 6, no. 4, pp. 41-47, Dec. 2006.

[21] G. B. Lee, J. Y. Ok, and E. G. Lim, "Method of Signature Extraction and Selection for Ransomware Dynamic Analysis," *KIISE Transactions on Computing Practices (KTCP)*, vol. 24, no. 2, pp. 99-104, Feb. 2018.

[22] D. Maimon and E. R. Louderback, "Cyber-dependent crimes: An interdisciplinary review," *Annual Review of Criminology*, vol. 2, pp. 191-216, 2019.

[23] Korea Internet & Security Agency (KISA) Internet Protect World & KrCERT, "What is Smishing?," [Online]. Available: https://www.boho.or.kr/cyber/smishing.do.

[24] S. M. Choi, "Cyber threats analysis using machine learning," M.S. thesis, Dept. CSE. Kor., Han-yang Univ., Seoul, Korea, 2020.

[25] Y. Kim and S. Chang, "A Hybrid Approach of Using Both Simulation plus Neural Networks for Window Design Optimization and HVAC Energy Consumption Prediction Modeling," *International Journal of Structural and Civil Engineering Research*, vol. 8, no. 4, pp. 300-309, Nov. 2019.

[26] B. Park, I. Yoo, J. Lee, S. Jang, S. Y. Kim, and Y. Kim, "A Reference Frame Selection Method Using RGB Vector and Object Feature Information of Immersive 360° Media," *Journal of IKEEE*, vol. 24, no. 4, pp. 1050-1057, 2020.

[27] J. M. Koo, S. D. Na, J. H. Cho, and M. N. Kim, "Melanoma Classification Algorithm using Gray-level Conversion Matrix Feature and Support Vector Machine," *Journal of Korea Multimedia Society*, vol. 21, no. 2, pp. 130-137, 2018.

[28] Y. H. Jo, "Early ransomware detection using machine learning," M.S. thesis, Dept. CSE. Kor., Kook-Min Univ., Seoul, Korea, 2020.

[29] G. U. Park and I. Jung, "Comparison of resampling methods for dealing with imbalanced data in binary classification problem," *The Korean Journal of Applied Statistics*, vol. 32, no. 3, pp. 349-374, 2019.

[30] Y. Lei, B. Yang, X. Jiang, F. Jia, N. Li, and A. K. Nandi, "Applications of machine learning to machine fault diagnosis: A review and roadmap," *Mechanical Systems and Signal Processing*, vol. 138, pp. 1-39, 2020.

[31] A. Arora, S. K. Peddoju, and M. Conti, "Permpair: Android malware detection using permission pairs," *IEEE Transactions on Information Forensics and Security*, vol. 15, pp. 1968-1982, 2019.

[32] J. M. Jung, H. J. Kim, S. J. Cho, S. C. Han, and K. W. Suh, "Efficient Android Malware Detection Using API Rank and Machine Learning," *Journal of Internet Services and Information Security*, vol. 9, no. 1, pp. 48-59. Feb. 2019.

[33] S. H. Park, M. Y. Kang, J. H. Park, S. J. Cho, and S. C. Han, "Analyzing the Effects of API Calls in Android Malware Detection Using Machine Learning," *Journal of Korea Institute of Information Security & Cryptology*, vol. 48, no. 3, pp. 257-263, Mar. 2021.

[34] Z. Ma, H. Ge, Y. Liu, M. Zhao, and J. Ma, "A combination method for android malware detection based on control flow graphs and machine learning algorithms," *IEEE access*, vol. 7, pp. 21235-21245, 2019.

[35] S. I. Imtiaz, S. U. Rehman, A. R. Javed, Z Jalil, X. Liu, and W. S. Alnumay, "Deep AMD: Detection and identification of Android malware using high-efficient Deep Artificial Neural Network," *Future Generation Computer Systems*, vo1. 115, pp. 844–856, Feb. 2021.