IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received January 29, 2020, accepted February 25, 2020, date of publication March 9, 2020, date of current version March 18, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2979477

Risk Assessment Scheme for Mobile Applications Based on Tree Boosting

KICHANG KIM, JINSUNG KIM, EUNBYEOL KO, AND JEONG HYUN YI
School of Software, Soongsil University, Seoul 06978, South Korea

Corresponding author: Jeong Hyun Yi (jhyi@ssu.ac.kr)

This work was supported in part by the Institute for Information and communications Technology Promotion (IITP) Grant funded by the Korea Government (MSIT) (Automatic Deep Malware Analysis Technology for Cyber Threat Intelligence) under Grant 2017-0-00168, and in part by the Global Research Laboratory (GRL) Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Science, ICT, and Future Planning under Grant NRF-2014K1A1A2043029.

ABSTRACT In the forthcoming era of IoT, where everything will be connected, mobile devices will play a key role in providing data sharing and user-centric services between devices. In such a service environment, if a mobile application is vulnerable to security threats and exposed to malicious behavior, malware can spread to hundreds of millions of connected devices. In particular, it is important to isolate and respond quickly to malicious mobile code. This requires the prediction of malicious behavior. Currently, security risk assessment schemes based on the permission use the description of the application or user review, but these schemes mostly offer a subjective evaluation, which inevitably reduces accuracy. In this paper, we thus propose a scheme for assessing security risk of Android mobile applications by analyzing their application programming interfaces (APIs) using machine learning. The key idea of the proposed scheme is to extract the APIs from the execution code of the application with reverse engineering analysis, such that each API can be compared with the malicious API database built from the existing malware dataset. Instead of simply judging the applications as malicious or benign, our scheme shows their risk as a score. To do this quantitative evaluation, we use an ensemble of tree boosting machine learning algorithms. To prove the practicality of the proposed scheme, we experiment with a set of benign and malicious real world samples, and compare our results with existing schemes. Experimental results show better performance and accuracy than conventional schemes based on Naive Bayes and simple ensemble algorithms. Our proposed scheme is expected to significantly contribute in responding rapidly to ever-more-intelligent malware of the future.

INDEX TERMS Malware detection, machine learning, XGBoost, risk assessment.

I. INTRODUCTION

The mobile application market is growing at a fast pace and so is the scale of mobile malware. According to McAfee's report, the number of mobile malware that reached 10 million in the first quarter of 2016 has increased to approximately 31 million in the fourth quarter of 2018 [7]. The largest share of devices affected by these mobile malware run on Android. Not only is the Android applications' market share the highest, but because of the open market policy of Android applications, anyone can easily modify and redistribute them [17]. Even if it is not a malicious application, it may require unnecessary personal information. Consequently, users continue to suffer from damages such as personal information leakage and financial losses. Therefore, there is a need for a scheme to assess the security risk of an application quantitatively based on its actual behavior. Research to assess the risk of applications has been ongoing [11]. DroidRisk [28] uses a permission-based evaluation method, whereas WHYPER [23], evaluates based on the application's description. RiskMon [16] and AutoReb [20] evaluate user reviews to assess risk, and APK Vulnerability Identification System (AVIS) [18] does the same based on application programming interfaces (APIs). Among these, schemes based on permissions, descriptions, and user reviews clearly indicate limitations in accurately analyzing the application's actual behavior. Conversely, in the case of API-based evaluation schemes such as AVIS, it is possible to evaluate the actual operation of the application accurately by analyzing the APIs based on the actual functional

The associate editor coordinating the review of this manuscript and approving it for publication was Ilsun You.

characteristics of the application. Although the AVIS scheme is meaningful in the sense that it takes the initial approach to evaluate security risk based on APIs, its classification accuracy is not high. This problem exists because it uses machine learning algorithm that unsuitable with data features. For example, according to the experiment results discussed later in this paper, even though the Naver Dictionary application [5] is a commonly used benign application, AVIS shows a false negative result by classifying it as malicious applications. To overcome the shortcomings of these existing schemes, we propose a scheme that overcomes the shortcomings of the existing scheme and greatly improves the accuracy by using the eXtreme Gradient Boosting (XGBoost) algorithm [9] for API classification, which is an essential part of the API-based risk assessment scheme.

Our work contributes to the existing literature by suggesting a detailed system design, and how the XGBoost is applied in the learning and decision phases for risk assessment of Android applications. It also contributes by providing a comparative analysis of the accuracy of the existing schemes through various experiments on benign and malware samples such as ransomware, adware, trojan, and spyware.

This paper is organized as follows. In Section II, we review the existing risk assessment schemes for Android applications. Section III summarizes the background of the XGBoost algorithm used in this work. Section IV describes in detail the proposed scheme. In Section V-C, we implement the proposed scheme and present the experimental results by evaluating security risk of an application against the actual dataset. Finally, we conclude the paper in Section VI.

II. RELATED WORKS

This section discusses existing security risk assessment schemes for Android mobile applications.

A. PERMISSION-BASED SCHEMES

These schemes evaluate the security risk of an application by analyzing the permissions that are requested by the target application and create a risk score by comparing the requested permissions to the predefined risk ranking. Representative schemes include DroidRisk and APKAuditor [27]. As these schemes depend on permissions, developers need to know which permissions are important for their applications. However, this evaluation method may not be accurate because application developers often require unnecessary permissions without knowing their exact meaning and/or necessity.

B. DESCRIPTION-BASED SCHEMES

These schemes measure the security risk of an application by analyzing the description of the application to infer what permissions are essential to run the application. Representative schemes include WHYPER and ADROIT [21]. Here, rather than simply looking at the requested permissions, the risk is assessed by comparing the permissions that are deemed necessary with regard to the execution of the application and the permissions that are actually requested, resulting in better accuracy than the aforementioned permission-based risk assessment schemes. However, because of the descriptions are written entirely by the developers themselves, there is a limit to accepting them as objective evaluation results.

C. USER REVIEW-BASED SCHEMES

Another approach of risk assessment based on user reviews is taken by RiskMon and AutoReb. These schemes evaluate security risk by analyzing the user reviews of the application and comparing the application's expected behavior to the actual behavior. However, as the assessment is based on a general user's review, an operation outside the function actually used by the user cannot be reflected in the evaluation and the probability of a reliable objective review is limited.

D. API-BASED SCHEMES

Recently, various schemes for analyzing security risk according to the sensitivity of the API used in the application have been proposed. By using APIs that actually function in the application, we can analyze the actual and accurate behavior and function of the application. If the API applied to the target application is also used in a large number of malicious applications, the target application is considered to be at high risk. Hence, an objective statistical analysis is possible here, which is why we propose a quantitative scheme based on this approach.

1) CHO's SCHEME

Cho's Scheme [10] is a quantitative assessment of the obfuscation techniques applied to protect the functionality of an application. The scheme uses the Naive Bayes [12] classifier to classify sensitive APIs to be protected. In terms of accessing a text classification method characterized by API characteristic information, it can be said that this scheme is closer to analyzing the actual behavior of the application than previous evaluation methods based on description and permission. However, the Naive Bayes classification used in this scheme is not suitable for API classification because it assumes the independence of each characteristic information. In addition, because of the use of a single classifier, the classification result is calculated to either 0 or 1. For example, even if the difference between the probabilities of being a sensitive and non-sensitive APIs are very small, it is classified into a slightly larger probability category, thereby significantly distorting the final classification result.

2) APK VULNERABILITY IDENTIFICATION SYSTEM

To improve upon the shortcomings of Cho's scheme, AVIS was introduced, which uses bagging ensemble that utilizes 10 classification algorithms, including Naive Bayes and decision tree [24]. The scheme combines these 10 different algorithms to analyze data from various perspectives and uses the average of all results to produce more stable results than Cho's scheme. However, the disadvantage is that the 10 algorithms are applied without flexibly configuring parameters according to the feature of the target data.

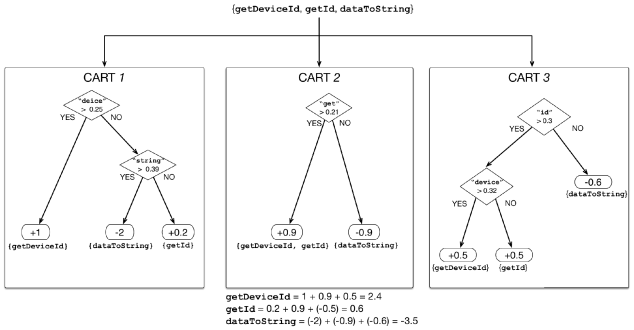


FIGURE 1. Example of CART prediction in XGBoost.

III. BACKGROUND

This section describes the XGBoost algorithm used in the proposed scheme and explains the reasons for its adoption.

A. XGBoost

XGBoost is an evolution of AdaBoost [13] and gradient boost [14], among decision tree-based boosting algorithms. Notably, it has the advantage of using a Classification and Regression Trees (CART) model [26] that allows classification and regression as well as the use of gradient descent to minimize errors. XGBoost also has the advantage of being able to compute complex calculations quickly using parallel and distributed processing.

1) CART

CART is a tree model used when making an ensemble model in XGBoost. In general, CART extends branch based on Gini impurity value, but CART in XGBoost makes a tree using gain value after selecting the root node (see subsection III-A.3). In this case, boosting is an algorithm of predicting accuracy by combining several simple classifiers, so that the depth variable is adjusted, and the tree is not deepened. Then, we update the weights to minimize the *Obj* value and proceed with tree boosting (see section III-A.2). Unlike most decision trees, which make only 0 or 1 decisions at leaf nodes, CART has a constant value of not only 0 or 1 leaf nodes but also weights. As a result, even models having the same classification result can be compared with superiority through constant values, thereby allowing a more sophisticated result analysis. Figure 1 illustrates how CART calculates each API's score after being generated by XGBoost when using the APIs the as training dataset. For example, suppose that x CARTs are generated by learning the XGBoost algorithm for a training dataset. In this case, in the API test dataset consisting of three APIs, {`getDeviceId`, `getId`, `dataToString`}, `getDeviceId` receives a leaf node value of 1, 0.9, and 0.5 by CART 1, CART 2, and CART 3, respectively. Next, we use $1 + 0.9 + 0.5 = 2.4$, which is the sum of the leaf node values of each CART, as the score for `getDeviceId`. Similarly, `getId` receives the leaf node values of 0.2, 0.9, and -0.5 in each CART and the score becomes $0.2 + 0.9 + (-0.5) = 0.6$; the `dataTostring` score becomes $(-2) + (-0.9) + (-0.6) = -3.5$. If the calculated API score is positive, it is classified as a *malicious API*, whereas if it is negative, it is classified as a *benign API*.

2) GRADIENT BOOSTING

To evaluate risk using XGBoost, the aforementioned CART has to be generated through pre-learning. After dividing the API provided by the training dataset into tokens, CART is constructed by receiving quantified values. Specifically, the performance of CART is measured by calculating the size of the error, not the accuracy of the tree, using the *objective* function. The objective function consists of regularization, which represents training loss and the complexity of the tree, and finds a combination of trees that maximizes performance by obtaining the optimal weight to minimize the objective function value. At this time, the gradient descent method is used to calculate the minimum error value using the slope of the function, which is much better than the simple error calculation method used in AdaBoost. In XGBoost, given the labeled data y_i, the predicted value $\hat{y}_i$ and its objective function *Obj* are calculated as follows:

$$\hat{y}_i = \sum_{s=1}^{S} f_s(x_i), \quad f_s \in \mathcal{F} \tag{1}$$

$$Obj = \sum_{i=1}^{n} L(y_i, \hat{y}_i) + \sum_{s=1}^{S} \Omega(f_s) \tag{2}$$

where S indicates the number of different tree models, $f_s()$ means the CART function generated in round s, x_i means the training data, and $\mathcal{F}$ does the space of trees. Also, $L()$ represents training loss and $\Omega()$ represents tree complexity, i.e., an XGBoost process of finding weights that minimize the loss function and tree complexity. In this case, the predicted value in round t may be expressed as $\hat{y}_i^{(t)} = \hat{y}_i^{(t-1)} + f_t(x_i)$ so the objective function is as follows:

$$Obj = \sum_{i=1}^{n} L(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) + \Omega(f_t) + constant \tag{3}$$

Now, as the round passes, training loss L is expanded until $t = 1$ to find the weight that minimizes the error of the model. For example, if we define training loss as square loss, we can expand as follows:

$$\begin{aligned} Obj^{(t)} &= \sum_{i=1}^{n} (y_i - (\hat{y}_i^{(t-1)} + f_t(x_i)))^2 + \Omega(f_t) + const \\ &= \sum_{i=1}^{n} [2(\hat{y}_i^{(t-1)} - y_i) f_t(x_i) + f_t(x_i)^2] + \Omega(f_t) + const \end{aligned} \tag{4}$$

Then $2(\hat{y}_i^{(t-1)} - y_i)$ becomes training loss in $t-1$ rounds. Because the formula of t rounds becomes the formula of $t-1$ rounds, it is possible to expand until $t = 1$ rounds. To do this, we expand the formula using the Taylor series. Prior to this,

we define $g_i = \partial_{\hat{y}^{(t-1)}} L(y_i, \hat{y}^{(t-1)})$, $h_i = \partial^2_{\hat{y}^{(t-1)}} L(y_i, \hat{y}^{(t-1)})$.

$$Obj^{(t)} \cong \sum_{i=1}^{n} [L(y_i, \hat{y}_i^{(t-1)} + g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i))] + \Omega(f_t) + constant \quad (5)$$

After excluding terms that are considered constants at time t, the following equation remains:

$$Obj^{(t)} \cong \sum_{i=1}^{n} [g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)] + \Omega(f_t) \quad (6)$$

Because the functions g and h are derivatives at time point $t-1$, they can be expanded until $t = 1$ rounds. Looking at the $\Omega()$ function where γ is the number of leaves and w is the leaf score, XGBoost defines it for a tree T as follows:

$$\Omega(f_t) = \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2 \quad (7)$$

Eventually, these two parameters will determine the complexity of the model. Now, organizing the objective function for w-s produces the following equation:

$$Obj^{(t)} \cong \sum_{j=1}^{T} [(\sum_{i \in I_j} g_j) w_j + \frac{1}{2}(\sum_{i \in I_j} h_i + \lambda) w_j^2] + \gamma T \quad (8)$$

where I_j means the instance set of leaf j. We now have a quadratic equation for w, so we can find w in which the value of the objective function is minimal. Therefore, the value of the optimum weight w^* and minimum of the objective function at that time are as follows:

$$w_j^* = -\frac{G_j}{H_j + \lambda}, \quad Obj = -\frac{1}{2} \sum_{j=1}^{T} \frac{G_j^2}{H_j + \lambda} + \gamma T \quad (9)$$

where $G_j = \sum_{i \in I_j} g_i$ and $H_j = \sum_{i \in I_j} h_i$. Gradually applying the weights calculated for a tree T to the next CART to minimize the error is called *gradient boosting*.

3) SPLIT FINDING

Gradient boosting allows you to calculate the performance of the tree at every step, so by infinitely increasing the branch of the tree, you will find the tree structure with the best performance. Theoretically, it would be ideal to make and compare all the tree combinations, but as that is not practical, we use a somewhat *greedy* algorithm to calculate the information gain. At every stage of pruning, the information gain is calculated as follows:

$$Gain = \frac{G_L^2}{H_L + \lambda} + \frac{G_R^2}{H_R + \lambda} - \frac{(G_L + G_R)^2}{H_L + H_R + \lambda} - \gamma \quad (10)$$

where L means left branch score and R means right branch score. Now make a myriad of trees, and calculate the information gain at the time of splitting the tree branch. Then, if the score is less than 0, you can split the branches to avoid *overfitting* and get a T tree combination model that achieves the best performance.

TABLE 1. Feature comparison of API based schemes.

	Cho's Scheme	AVIS	Proposed Scheme
Underlying Techniques	Naive Bayes	Bagging ensemble	XGBoost and bagging ensemble
Number of Classifiers	Single classifier	10 different classifiers	Multiple classifiers
Training Dataset build-up	Complete training set	Random sampling with replacement	Random sampling with replacement over weighted data
Weighted mechanism provided	No	No	High weight to errors
Decision basis information	Binary estimate	Simple average	Weightetd average
Advantages	Easy to use Fast learning time	Stability	High accuracy Prevent overfitting Prevent underfitting
Disadvantages	Low accuracy	Underfitting	Slow learning time

B. FEATURE COMPARISON

Now, based on our analysis of API-based risk assessment schemes described in Section II-D, we will examine the reasons for using XGBoost in this paper. First, Cho's scheme, although using a simple Naive Bayes classifier, is somewhat less accurate. However, it is a significant contribution in the sense that it was the first to present a way to evaluate an application based on an API. In the case of AVIS, it can be said that it is good to try to prevent *overfitting* by classifying the training dataset by the restorative extraction method using ten different algorithms. However, the weighting mechanism is not applied individually, and as the simple average value is determined as the final score during classification, there is a limit to reducing errors and increasing accuracy. For this reason, we utilize XGBoost. The XGBoost algorithm not only randomly samples the training dataset but also recalculates the weight at each step, thus preventing *overfitting* and *underfitting*. In addition, the general boosting algorithm has a disadvantage in that the performance of the model is slightly lower than the bagging that can be processed in parallel because the classifiers are generated sequentially. However, because XGBoost uses the parallel and distributed processing method, the learning and classification speeds are also fast. Table 1 compares the proposed scheme with Cho's scheme and AVIS.

IV. PROPOSED SCHEME

The proposed scheme has been divided into a *preprocessing* phase, a *learning* phase, and a *decision* phase. In the preprocessing phase, an API is extracted from benign and malicious samples to construct an initial unclassified training dataset. In the learning phase, a trained database is constructed from an unclassified training dataset using XGBoost algorithm. In decision phase, upon input of a target application, it is determined whether the target application is malicious or not. The overall structure of the proposed scheme is shown in Figure 2.

A. PREPROCESSING PHASE

1) API EXTRACTION

To use the data for learning, we analyze, extract, and label the APIs commonly used in benign and malicious applications

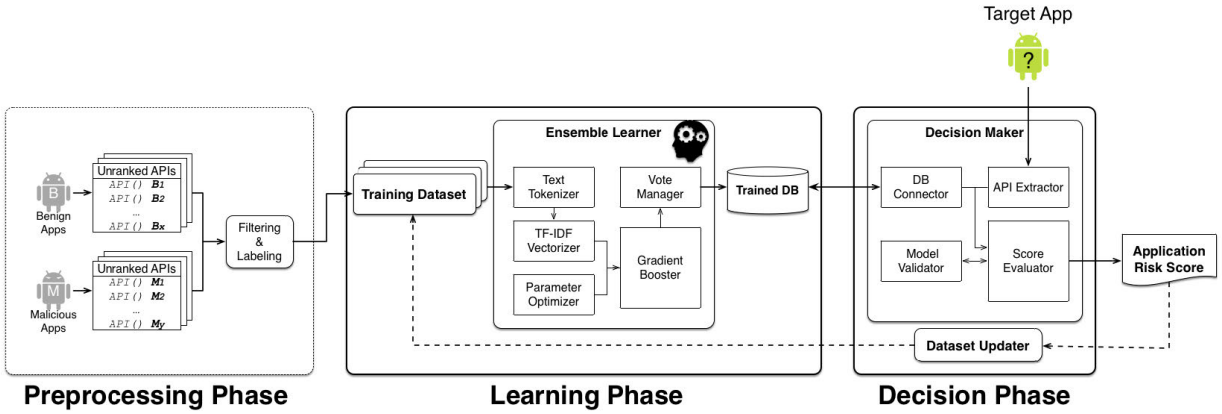


FIGURE 2. Overall architecture of proposed scheme.

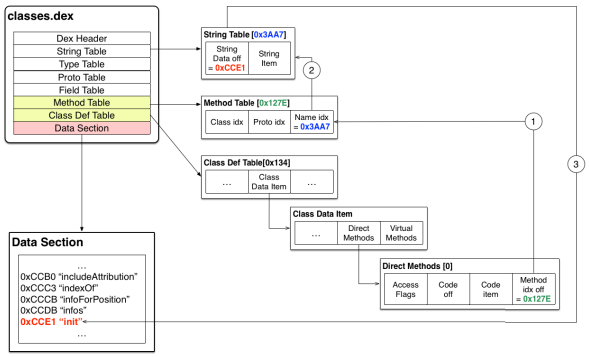


FIGURE 3. How to extract APIs from `.dex`.

to generate the training dataset. To extract the APIs used in an application, we first analyze the **`classes.dex`** file that contains the code related to an Android application's functionality. Figure 3 shows the structure of the **`.dex`** file of the application [3] and the API extraction method.

The **`.dex`** file consists of eight fields, and it can be seen that **`method`**-related data are included in the `Method Table` and the `Class Def Table`. The `Class Def Table` is a list of user-defined classes. It contains **`class`** information such as `Class Data Item`. `Direct` and `Virtual Methods` have information about each **`method`** in the **`class`**. `Direct Method` contains information indicating the offset of the method, which indicates the index of the `Method Table`. Finally, when reaching the `String Table` through the `Name index` value obtained by calculating the difference between the `Method index` offsets of the `Direct` and `Virtual Methods` in the `Method Table`, it is possible to find and extract the **`API name`** in the `Data Section`. After the API is extracted, static analysis [1] is used to generate a list of commonly used APIs.

2) FILTERING AND LABELING

After extraction, the APIs are filtered according to whether the frequency of occurrence in two different populations exceeds a given threshold. Specifically, we define $m-thd$ as

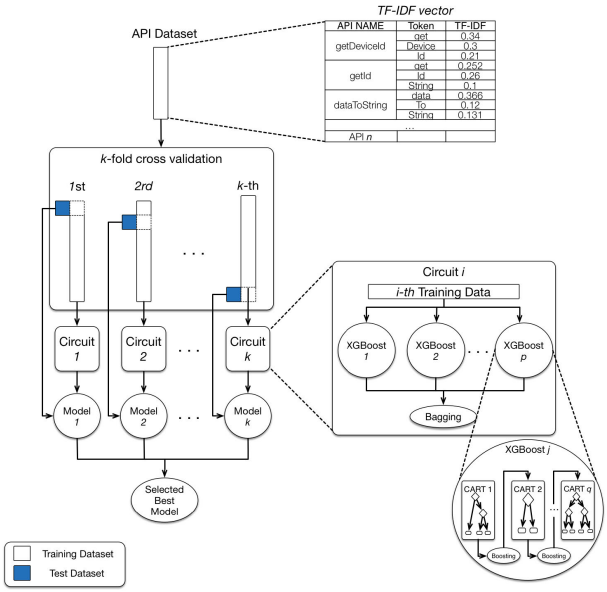


FIGURE 4. Learning phase structure.

the threshold in *malicious* samples, $b-thd$ as the threshold in *benign* samples, $m-value$ as the frequency of occurrence of the API in *malicious* samples, and $b-value$ as the frequency of occurrence in *benign* samples. Then, if $m-value \geq m-thd$ and $b-value < b-thd$, the API is classified as a ***sensitive API***, and labeled with a value of 1. Conversely, $m-value < m-thd$ and $b-value \geq b-thd$, the API is classified as a ***non-sensitive API***, and labeled with a value of 0.

B. LEARNING PHASE

After the preprocessing phase, the API is vectorized in the learning phase. We then generate an ensemble model using k-fold cross validation [19] and select the best model to determine the API's ranking. A detailed structure of the learning phase is shown in Figure 4.

1) API TOKENIZING

As shown in Figure 5, when the training and test dataset are prepared, the API is divided into tokens

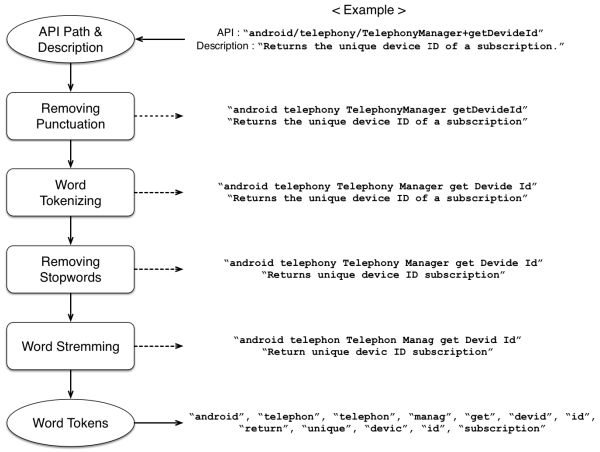


FIGURE 5. Word tokening procedure of API information.

into units of words, which is the input form of the ensemble learner, to generate the API ranking through the ensemble learner. For example, given the API path "`android/telephony/TelephonyManager+getDeviceId`", it will go through *Removing Punctuation, Word Tokenizing, Removing Stopwords, Word Stemming*, and will finally decompose into six word tokens: `telephon`, `telephon`, `manag`, `get`, `devic`, and `id`. In addition, the API description, as well as the API path, are entered as learner input. In this case, *Removing Stopwords* is used to remove meaningless articles or periods. For example, we decompose the API description `Returns the unique device ID of a subscription` into five tokens: `returns`, `unique`, `device`, `id`, and `subscription`.

2) TF-IDF VECTORIZATION

Word tokens are vectorized using the Term Frequency-Inverse Document Frequency (TF-IDF) [8] vectorizer. TF-IDF is a weight used for character string analysis and indicates how important a word is in the document and the entire document group. Term Frequency (TF) is a value that indicates how often a particular word appears in the document. In general, if the word has a high TF value, it is judged to be an important word in the document. However, if the word appears in almost all documents included in the entire document group, it is difficult to determine whether the word can represent the characteristics of the document. This is called Document Frequency (DF), and TF-IDF is a product of TF and Inverse DF (IDF). Therefore, a word having a high TF-IDF value indicates an important word. In equation 11, t represents a specific word and d represents a document, that is, property information of an API such as Table 2. D represents the entire document set, that is, the entire property information set of all APIs. $f(t, d)$ is a function for finding the total frequency of the word t in document d. As the frequency can be too large for $f(t, d)$, we use the augmented frequency to adjust this. Augmented frequency is fixed to a maximum of 1 so that it represents the relative frequency of words

TABLE 2. Property Information of "`getDeviceID`" API.

Package Name	`android.telephony`
Class Name	`telephonyManager`
Method Name	`getDeviceID`
Description	Returns the unique device ID of a subscription.

along the length of the document. The following equation is the augmented frequency TF-IDF calculation used in the proposed scheme:

$$\begin{aligned} tf(t,d) &= 0.5 + \frac{0.5 \times f(t,d)}{maxf(w,d) : w \in d} \\ idf(t,D) &= \log \frac{|D|}{|d \in D : t \in d|} \\ tf - idf(t,d,D) &= tf(t,d) \times idf(t,D) \end{aligned} \tag{11}$$

Once we have completed string vectorization, we are ready to train with the classifier. Now we are going to classify all the APIs with the machine learning classifier. The algorithm used for classification is based on the XGBoost algorithm described earlier. For example, assuming that the property information of the `getDeviceID`, API is given as shown in Table 2, the process of classifying the `getDeviceID` API through the XGBoost algorithm can be shown as in Figure 5.

When this property information passes through API Tokenizer and TF-IDF vectorizer, it is vectorized into 10 tokens such as {`android`: $x1$, `telephon`: $x2$, ..., `subscription`: $x10$}. At this time, the values of $x1$, $x2$, ..., $x10$ are TF-IDF values and are given differently for each other. After that, if the API is classified through each tree, the classification result for each tree is obtained according to the previously generated vector value. If the sum of all the values is positive, it is classified as a sensitive API; but if negative, the API is classified as a non-sensitive API.

3) PARAMETER OPTIMIZATION

Because the training dataset depends on the k value of k-fold cross validation, we also need to select the appropriate k value. To take advantage of the high accuracy and analysis power of the XGBoost algorithm, and the stability and "what can score the results" of the bagging ensemble algorithm, we use a combination of p XGBoost classifiers as the ensemble model. When generating the classifier, we proceed with parameter optimization to find the optimal parameter value for the classification. As the accuracy of the classifier varies greatly depending on the parameter value, we directly experiment with various parameter combinations to find the optimal parameter value. Representative parameters of the XGBoost classifier include `n_rounds`, which sets the maximum number of trees; `early_stopping_rounds`, which sets the minimum number of trees that can stop the expansion of the tree when the best results are obtained; and `max_depth`, which limit tree pruning. In the case of gradient boosting, it is important to find the optimal parameter through several experiments because if the size tree of the entire tree becomes too large, there is a risk of overfitting.

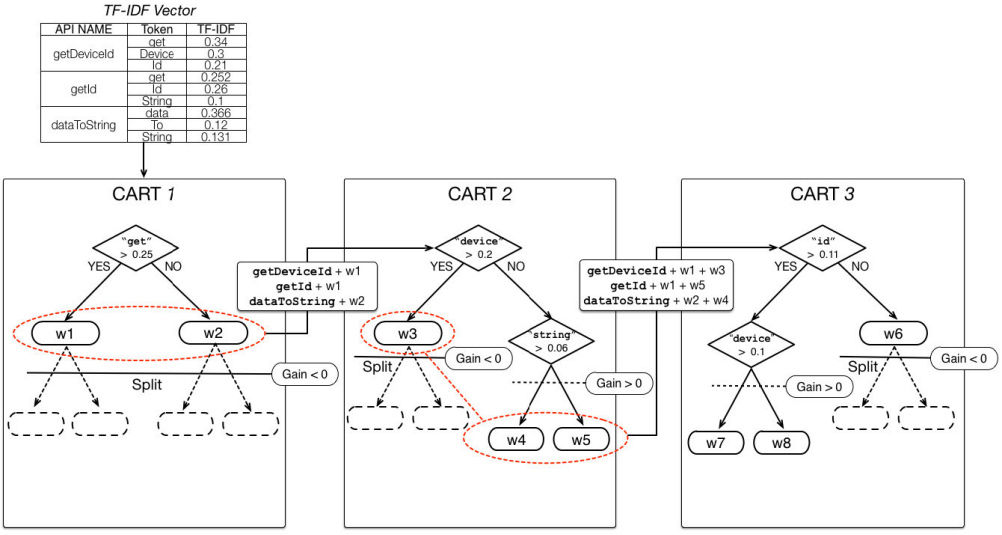


FIGURE 6. Tree boosting process of XGBoost.

4) GRADIENT BOOSTING

The p XGBoost classifiers generated after parameter optimization are combined as ensemble models. To generate the ensemble classifier, we randomly extract the training dataset and generate n training dataset. Next, we generate p XGBoosts in parallel using k training dataset through k-fold cross validation. Then, for boosting, we generate the first CART from the training dataset again and increase the number of trees up to q by calculating the tree performance with gradient boosting each round. When growth causes performance to deteriorate or the risk of overfitting is expected, the XGBoost algorithm stops growing the tree and generates an XGBoost classifier. Figure 6 shows the process of boosting three trees by calculating *Gain* (refer to subsection III-A.3) and *Obj* (refer to subsection III-A.2) when a training dataset is entered. When the training dataset is put in, the criteria are randomly set, and *CART* 1 is grown only while the *Gain* value is positive. If the *Gain* is negative, we do not divide the branch. We find w to minimize *Obj* and write it as the leaf node value. Then this value is passed to the input of the next CART. For example, for {`getDeviceId`, `getId`, `dataToString`}, the 3 training dataset are classified as {`getDeviceId`, `getId`} = $w1$, {`dataToString`} = $w2$ in *CART* 1. This leaf node values become weighted and are delivered with data to *CART* 2, as shown in Figure 6. With the received data, *CART* 2 also divides the branch until *Gain* becomes negative, sets the leaf node value through *Obj*, and sends it to *CART* 3. In this way, tree boosting is carried out by repeating *Gain* and *Obj* calculations for each round of tree generation and passing the weighted data to the next tree.

5) VOTE MANAGEMENT

After the k classifier is generated, we use k-fold test dataset to get the accuracy of k classifiers. We select the classifier with the highest accuracy as the final best model. If the API is classified through the selected best model, the API is scored using the average value of the total p classification results. For example, if 10 of the 10 XGBoost classifiers classify an API as sensitive, then a score of 1.0 is assigned to the API. If 7 of the 10 classifiers classify as sensitive, a score of 0.7 is assigned. Algorithm 1 shows the pseudocode of the ensemble learner described so far.

C. DECISION PHASE

After an API's risk ranking is generated through an ensemble learner, the decision maker evaluates the application's security risk by comparing the risk ranking to the APIs used by the target application. After the API extractor extracts the APIs from the target APK file and the database connector loads the API's ranking previously created by the ensemble learner, the score evaluator compares the extracted APIs with the API ranking list and assigns scores to each API. The number of APIs corresponding to scores between 0.0 and 1.0 is multiplied by each score and all the values are added together and divided by the total number of APIs to generate the final application risk score. We then update the input dataset by adding the newly generated scores from the target application to the training dataset.

V. EXPERIMENTS

This section describes our implementation and experimental results of the proposed scheme.

A. EXPERIMENTAL SETUP

The experiment environment consisted of Windows 10 Pro using Java Development Kit version 1.8.0 and Python 3.6.5. We use XGBoost algorithm, which provides the gradient boosting framework for API classification, `scikit`, a representative Python algorithm library, and `nltk` for natural language processing.

The initial training dataset was constructed using 2,700 benign samples, selected from the top 10% applications

in each category in Google Play Store [4], and 7,400 malicious samples, provided by VirusShare [6] and Contagio [2].

Algorithm 1 Pseudocode of the Proposed Ensemble Learner

Input: *API DataSet*
Output: *prediction*

load the APIs
Text tokenizing and TF − IDF vectorization
k − fold cross validation
(Divided into training and test dataset)
for $i := 1$ *to* k **do**
  $p =$ *number of XGBoosts*
  for $j := 1$ *to* p **do**
    $q =$ *Maximum number of CARTs* $(= round)$
    input Training DataSet into CART 1
    for $r := 1$ *to* q **do**
      learn the CART
      split finding
      if $Gain(r) < 0$ **then**
        split terminate
      end if
      boost the tree
      if $Obj(r) < Obj(r+1)$ **then**
        boosting terminate
      end if
    end for
    sum predictions
  end for
  mean predictions
  input Test Dataset into $i - th$ *XGBoost Model*
end for
Select Best Model
Make API ranking

B. CONFIGURATION

1) TRAINING DATASET LABELING

Each API's initial label in the initial training dataset was determined by experimenting with a target sample of benign and malicious applications. Figure 7 shows the number of APIs used in the benign and malicious application groups; it also analyzes the frequency of occurrences of each API in benign and malicious applications.

In the distribution, the $m - value$ on the X-axis represents the frequency of occurrence of the corresponding APIs in the malicious samples and the $b - value$ on the Y-axis represents the frequency of occurrence in the benign samples. In other words, the lower right shows the APIs mainly used for malicious applications and the upper left shows the APIs used only for benign applications. Among these, only the APIs that can be clearly classified as sensitive or non-sensitive APIs are used to generate labels. Through repeated experiments,

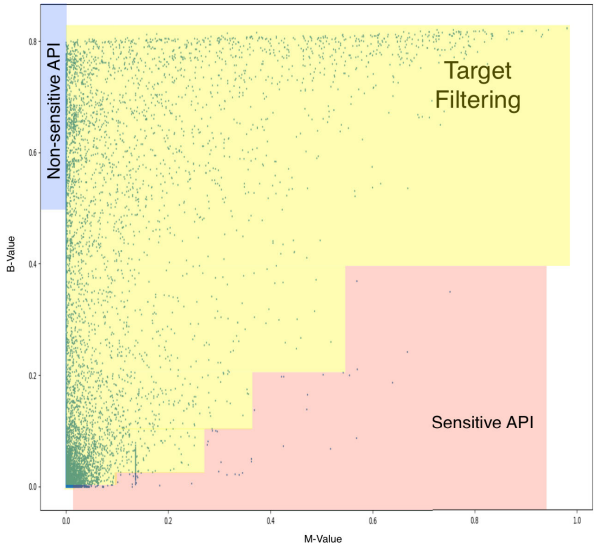


FIGURE 7. *m*-value and *b*-value distribution for initial training dataset.

TABLE 3. Accuracy by *k*-value.

k	3	5	8	**10**	15	20
Max Accuracy (%)	94	94	94	**95**	95	95
Average Accuracy (%)	94	93	94	**94**	94	94
Log Loss	0.18	0.17	0.18	**0.17**	0.17	0.17

the label generation conditions of the experiment showing the highest accuracy are as follows:

1) Sensitive API (`label:1`)
   - $m - value \geq 60\%$ and $b - value < 40\%$
   - $m - value \geq 40\%$ and $b - value < 20\%$
   - $m - value \geq 30\%$ and $b - value < 10\%$
   - $m - value \geq 10\%$ and $b - value < 1\%$
   - $m - value \geq 1\%$ and $b - value < 0.3\%$
   - $m - value \geq 0.3\%$ and $b - value = 0\%$
2) Non-sensitive API (`label:0`)
   - $m - value < 0.1\%$ and $b - value \geq 50\%$

2) CLASSIFIER MODELING

The classifier used for API classification goes through k-fold validation with the gradient tree boosting model provided by XGBoost algorithm. To find the optimal k value, we measured the accuracy of the classification model by setting k from 3 to 20 on $k = 10$ that had the highest accuracy in previously proven studies [15], [22], [25]. The measurement results are shown in Table 3. If k is too large, there may be underfitting and model generation takes a lot of time, so we use 10 as the most appropriate k value.

Because we decided to use $k = 10$, we divided the training dataset into 10 and made a classification model with 9 of them as the *training dataset*. Then, we measured the accuracy with the remaining one as the *test dataset*. After modeling 10 times (so that all 10 training dataset blocks are used once as test dataset) we measured the accuracy of the classification model and use the one that has the highest accuracy.

TABLE 4. Accuracy by XGBoost count.

p	Average Accuracy (%)	Log Loss
1	93.09	0.217
3	93.4	0.208
5	93.53	0.208
7	93.33	0.208
10	**93.45**	**0.207**
13	93.37	0.207
15	93.35	0.207
17	93.35	0.207
20	93.21	0.207

TABLE 5. Accuracy with 10-fold cross validation (%).

NB	AVIS	XGB	Proposed
79.79	83.68	90.88	85.42

TABLE 6. Classification time (seconds).

NB	AVIS	XGB	Proposed
217	958	344	1,681

We next analyze p XGBoost algorithms to find the appropriate number of XGBoost algorithms in the XGBoost ensemble model. Experimental results in Table 4 show that the log loss is low while the accuracy is high when 10 XGBoost algorithms are used. Therefore, we choose to ensemble 10 XGBoost algorithms.

The XGBoost algorithm has parameters such as `n_rounds`, `early_stopping_rounds`, `max_depth`, etc., as mentioned in Section IV-B.3. As we are experimenting with each parameter, we use the default values (`n_rounds=10, early_stopping_rounds=0, max_depth=6`) because it does not significantly affect model accuracy in this paper.

Tables 5 and 6 show the 10-fold cross-validation results and classification speed comparisons of a single Naive Bayes classifier (NB), Naive Bayes classifier with bagging ensemble (AVIS), a single XGBoost classifier (XGB), and our proposed scheme (an XGBoost classifier model with bagging ensemble). The XGBoost algorithm used in the proposed scheme is more accurate than the more-generally used Naive Bayes classifier. In addition, the XGBoost ensemble classifier that uses both bagging and boosting (proposed scheme) is not much different from the single XGBoost algorithm in terms of accuracy, but the classification speed is the slowest.

3) API RANKING

We use our classifier to classify over 30,000 APIs to create an API risk ranking. The API rankings created by implementing the proposed scheme are shown in Table 7. It can be seen that sensitive APIs related to SMS, permission access, device information, etc., are placed on top.

4) THRESHOLD ADJUSTMENT

Next, we can extract the APIs used by the target application and compare the extracted APIs with the API risk ranking list to assign an application risk score. Prior to the evaluation, the thresholds are adjusted to facilitate classification accuracy measurements. Figure 8 and Table 8 show the thresholds set to measure the classification accuracy of each classifier. The red color of the graph indicates the score distribution of malicious applications, while the blue color indicates the score distribution of benign applications. Based on the threshold, it is evident that the classification is better in the XGBoost-based classifier than in the Naive Bayes-based classifier.

TABLE 7. Ranking based on API analysis.

Rank	APIs
1.0	`SmsManager.sendTextMessage`, `SmsManager.sendDataMessage`, ···
0.9	`TelephonyManager.getDeviceId`, `AccessibilityService.findFocus`, `AccountManager.getAuthToken`, `File.setLastModified`, ···
0.8	`UsbDevice.getDeviceId`, `BasicHttpResponse.getStatusline`, `SendSmsResult.getSendStatus`, ···
0.7	`TelephonyManager.getLinelNumber`, `AppwidgetHost.deleteHost`, `HttpClientConnection.sendRequestHeader`, ···
0.6	`BufferedReader.ready`, `ServiceState.getRoaming`, `PhoneNumberUtils.isWellFormedSmsAddress`, ···
0.5	`RecyclerViewLayoutTest.testScrollStateForSmoothScroll`, `AccessibilityServiceInfo.getId`, `ParserAdapter.endDocument`, ···
0.4	`ContactsContract.Groups.newEntityIterator`, `MediaStore.Audio.Artists.Albums.getContentUri`, ···
0.3	`Loader.dataToString`, `ApplicationInfo.dump`, `PermissionInfo.writeToParcel`, `AssetManager.openFd`, ···
0.2	`TextUtils.split`, `XmlSerializer.cdsect`, `LangUtils.hashCode`, ···
0.1	`CharArrayBuffer.buffer`, `CharArrayBuffer.ensureCapacity`, ···
0.0	`EntityUtils.toString`, `Driver.parseSubTree`, ···

TABLE 8. Threshold value of each classifiers.

	NB	AVIS	XGB	Proposed
Threshold	0.4528	0.4541	0.6875	0.7095
Adjustment	+0.0472	+0.0459	-0.1875	-0.2095

TABLE 9. Assessment result with Naver Dic App.

NB	AVIS	XGB	Proposed
0.4687	0.512	0.4322	0.4635

C. EXPERIMENTAL RESULTS

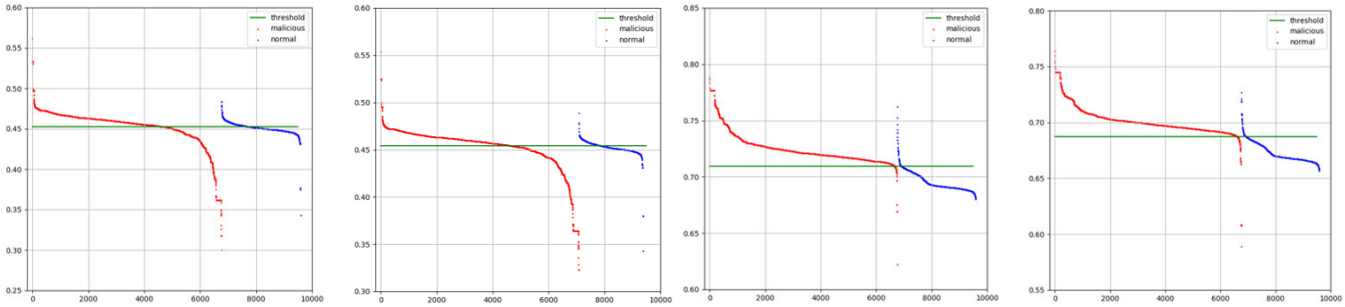
Once the threshold is adjusted, the target application is classified as a malicious application if its risk score is above 0.5 and a benign application if it is less than 0.5. Apart from the overall results, we discuss results for four specific types of applications as well.

1) CASE-1: NAVER DIC APPLICATION

Table 9 shows the results of evaluating the Naver Dictionary application (in the Top Downloads list of Google Play store for Korea) through the proposed scheme. The Naver Dictionary application includes APIs with relatively high-risk scores such as those that are account-related and SMS-related, so AVIS classified the application as *malicious* with a score of 0.5120. In contrast, it was classified as a *benign* application by the other three, that is Naive Bayes with a score of 0.4687, XGBoost with a score of 0.4322, and the proposed scheme with a score of 0.4635.

2) CASE-2: RANSOMWARE

Table 10 is a list of 47 application samples classified in Contagio as ransomware. Table 11 shows the accuracy of the



(a) Naive Bayes (b) AVIS (c) XGBoost (d) Proposed

FIGURE 8. Threshold value selection.

TABLE 10. Selected ransomware list.

No.	APK Name
1	sample.apk (ransomware locker)
2	Koler.C.apk
3	Android.Trojan.SLocker.DZ/Fake videoplayer/ Ransomware.apk
4	Contagio_ransom_23881.apk
5	Contagio_ransom_23882.apk
	...
47	ru.blogspot.playsib.savageknife.apk

TABLE 11. Assessment result with ransomware.

NB	AVIS	XGB	Proposed
0.64	0.67	0.86	1

TABLE 12. Selected adware list.

No.	APK Name
1	live.photo.savanna.apk
2	Contagio_adware_10981.apk
3	Contagio_adware_10982.apk
4	Contagio_adware_10994.apk
5	samples.apk (MobiDash)
	...
51	Airpush.apk

TABLE 13. Assessment result with adware.

NB	AVIS	XGB	Proposed
0.93	0.93	1	1

test results for each application sample. We can see that the classification accuracy for ransomware applications is higher in the proposed scheme than in the other three schemes.

3) CASE-3: ADWARE

Table 12 is a list of 51 application samples classified as adware in Contagio. Table 13 shows the accuracy of the test results for each application sample. It can be seen that Naive Bayes and AVIS are relatively inaccurate, whereas XGBoost and our proposed scheme classify applications more accurately.

4) CASE-4: TROJAN/SPYWARE

Table 14 is a list of 107 application samples classified as Trojan/Spyware in Contagio. The results of experimenting with each sample are shown in Table 15. The accuracy of NB was 0.33 (highest error rate) and AVIS was 0.5. Conversely, XGB and the proposed scheme showed high accuracy.

TABLE 14. Selected Trojan/Spyware list.

No.	APK Name
1	infostealer.apk (Android Tetus)
2	infostealer.apk (Assassins Creed)
3	infostealer.apk (Fakemart)
4	SMStrojan.apk (Fakemart)
5	SMStrojan.apk (MSZombie.A)
	...
107	Trogoogle.apk

TABLE 15. Assessment result with Trojan/Spyware.

NB	AVIS	XGB	Proposed
0.33	0.5	0.94	0.95

TABLE 16. Assessment result for all APKs.

	NB	AVIS	XGB	Proposed
Accuracy (TP+TN) / (TP+TN+FP+FN)	0.68	0.70	0.97	0.98
True Positive Rate TP / (TP+FN)	0.68	0.71	0.98	0.99
False Positive Rate FP / (FP+TN)	0.32	0.30	0.06	0.05

We acknowledge that as there are less than 50 items in each Contagio dataset, the experiments cannot be considered accurate. Nevertheless, we can see that the proposed scheme correctly classifies almost all applications.

5) OVERALL ACCURACY

Table 16 shows the classification accuracy for all applications. Because of classification, NB did not show a meaningful classification result because of the high false positive rate. AVIS is more accurate than NB, but there is a problem in that misclassification occurs for large benign applications. Conversely, XGB and the proposed scheme classify with very high accuracy. In particular, the proposed scheme shows a much higher accuracy than AVIS in benign and malicious applications.

TABLE 17. Risk score of benign samples.

APK Name	Risk score
co.kr.daisomall	0.4591
Coloring_com.forqan.tech.PrincessColoring	0.460807
com.abyz.ezphoto	0.46208
com.actionsmicro.ezcast	0.46311
com.adobe.adobephotoshopfix	0.456303
...	
Qualcomm Lte Broadcast SDK	0.501881

The proposed scheme uses both bagging and boosting, making it the slowest in terms of classification speed. However, we do not mind that this is a big issue because it represents the speed of making API ranking lists and not the classification speed of applications. Rather, the bigger issue is the classification of malicious applications as benign, which can put the user's device at a serious security risk. However, the proposed scheme could be of the value of research in that the accuracy of the classification of malicious applications has been considerably increased.

Table 17 shows the risk score for benign applications. An application called `Qualcomm Lte Broadcast (QLB) SDK` is a benign application commonly used but has been classified as a malicious application in all four schemes, including the proposed scheme. However, because this application has many functions that can access location-related, account-related, and network-related personal information to provide `LTE Broadcast Service`, it can actually become a personal information leakage path. Based on the above, it may seem that the benign application is not detected properly. However, this is just a special case and the proposed scheme is 99.07% accurate when tested against 2,700 benign applications.

Therefore, even in benign application, the user can be alerted by objectively determining the actual risks and classifying them as malicious applications. However, as the difference between the average scores of benign and malicious applications is not so large, it is necessary to make a clear distinction by improving the risk scoring scheme.

VI. CONCLUSION

With the growth of the mobile applications market, the damage caused by malicious applications targeting vulnerabilities in Android applications is increasing daily. Moreover, due to the nature of Android, where it is easy to obtain and redistribute code, general users may not even notice that their personal information is leaked. In particular, not only malicious applications but some commonly used applications also collect personal and sensitive information unnecessarily. Hence, an objective evaluation scheme is needed, not just for the classification of malicious applications but for benign applications as well. Conventional application evaluation schemes, which are based on application description or permissions, are not suitable for analyzing actual application behavior. However, API-based evaluation schemes, which have been studied recently, need to be developed further because they reflect the actual operation of the application. In this paper, we demonstrated the possibility of a better application security risk assessment than conventional API-based schemes using an XGBoost-based machine learning algorithm that can be easily tuned after learning. In the future, we expect to increase performance by increasing training sets or performing additional experiments with more deep learning algorithms.

REFERENCES

[1] *Apktool*. Accessed: Nov. 2019. [Online]. Available: https://ibotpeaches.github.io/Apktool/

[2] *Contagio*. Accessed: Nov. 2019. [Online]. Available: http://contagiodump.blogspot.kr/

[3] *Dalvik Executable Format*. Accessed: Nov. 2019. [Online]. Available: http://source.android.com/devices/tech/dalvik/dex-format.html

[4] *Google Play Store*. Accessed: Nov. 2019. [Online]. Available: https://play.google.com/store/apps

[5] *Naver Dictionary Application*. Accessed: Nov. 2019. [Online]. Available: https://play.google.com/store/apps/details?id=com.nhn.android.naverdic

[6] *VirusShare*. Accessed: Nov. 2019. [Online]. Available: https://virusshare.com/

[7] McAfee Labs. *McAfee Mobile Threat Report*. Accessed: Nov. 2019. [Online]. Available: https://www.mcafee.com/enterprise/en-us/assets/reports/rp-mobile-threat-report-2019.pdf

[8] A. Aizawa, "An information-theoretic perspective of tf–idf measures," *Inf. Process. Manage.*, vol. 39, no. 1, pp. 45–65, 2003.

[9] T. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system," in *Proc. 22nd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining (KDD)*, 2016, pp. 785–794.

[10] T. Cho, H. Kim, and J. H. Yi, "Security assessment of code obfuscation based on dynamic monitoring in Android things," *IEEE Access*, vol. 5, pp. 6361–6371, 2017.

[11] A. P. Felt, E. Ha, S. Egelman, A. Haney, E. Chin, and D. Wagner, "Android permissions: User attention, comprehension, and behavior," in *Proc. Symp. Usable Privacy Secur.*, 2012, pp. 1–14.

[12] E. Frank and R. R. Bouckaert, "Naive Bayes for text classification with unbalanced classes," in *Proc. Eur. Conf. Princ. Data Mining Knowl. Discovery*. Cham, Switzerland: Springer, 2006, pp. 503–510.

[13] Y. Freund and R. E. Schapire, "Experiments with a new boosting algorithm," in *Proc. Int. Conf. Mach. Learn.*, vol. 96, 1996, pp. 148–156.

[14] J. H. Friedman, "Greedy function approximation: A gradient boosting machine," *Ann. Statist.*, vol. 29, no. 5, pp. 1189–1232, Oct. 2001.

[15] G. James, D. Witten, T. Hastie, and R. Tibshirani, *An Introduction to Statistical Learning: With Applications in R*. Berlin, Germany: Springer, 2013.

[16] Y. Jing, G.-J. Ahn, Z. Zhao, and H. Hu, "Riskmon: Continuous and automated risk assessment of mobile applications," in *Proc. ACM Conf. Data Appl. Secur. Privacy*, 2014, pp. 99–110.

[17] J.-H. Jung, J. Y. Kim, H.-C. Lee, and J. H. Yi, "Repackaging attack on Android banking applications and its countermeasures," *Wireless Pers. Commun.*, vol. 73, no. 4, pp. 1421–1437, Dec. 2013.

[18] H. Kim, T. Cho, G.-J. Ahn, and J. H. Yi, "Risk assessment of mobile applications based on machine learned malware dataset," *Multimedia Tools Appl.*, vol. 77, no. 4, pp. 5027–5042, Feb. 2018.

[19] R. Kohavi, "A study of cross-validation and bootstrap for accuracy estimation and model selection," in *Proc. Int. Joint Conf. Artif. Intell.*, Montreal, QC, Canada, vol. 2, 1995, pp. 1137–1145.

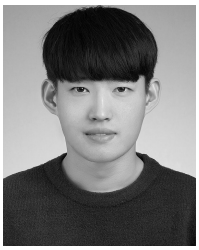
[20] D. Kong, L. Cen, and H. Jin, "AUTOREB: Automatically understanding the review-to-behavior fidelity in Android applications," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2015, pp. 530–541.

[21] A. Martin, A. Calleja, H. D. Menendez, J. Tapiador, and D. Camacho, "ADROIT: Android malware detection using meta-information," in *Proc. IEEE Symp. Ser. Comput. Intell. (SSCI)*, Dec. 2016, pp. 1–8.

[22] J. G. Moreno-Torres, J. A. Saez, and F. Herrera, "Study on the impact of partition-induced dataset shift on k-fold cross-validation," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 23, no. 8, pp. 1304–1312, Aug. 2012.

[23] R. Pandita, X. Xiao, W. Yang, W. Enck, and T. Xie, "WHYPER: Towards automating risk assessment of mobile applications," in *Proc. USENIX Secur. Symp.* Washington, DC, USA: USENIX, 2013, pp. 527–542. [Online]. Available: https://www.usenix.org/conference/usenixsecurity13/technical-sessions/presentation/pandita

[24] J. R. Quinlan, "Induction of decision trees," *Mach. Learn.*, vol. 1, no. 1, pp. 81–106, 1986.
[25] J. D. Rodriguez, A. Perez, and J. A. Lozano, "Sensitivity analysis of k-Fold cross validation in prediction error estimation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 32, no. 3, pp. 569–575, Mar. 2010.
[26] C. D. Sutton, "Classification and regression trees, bagging, and boosting," *Handbook Statist.*, vol. 24, pp. 303–329, Jan. 2005.
[27] K. A. Talha, D. I. Alper, and C. Aydin, "APK auditor: Permission-based Android malware detection system," *Digit. Invest., Int. J. Digit. Forensics Incident Response*, vol. 13, pp. 1–14, Jun. 2015.
[28] Y. Wang, J. Zheng, C. Sun, and S. Mukkamala, "Quantitative security risk assessment of Android permissions and applications," in *Proc. IFIP Annu. Conf. Data Appl. Secur. Privacy (IFIP)*. Cham, Switzerland: Springer, 2013, pp. 226–241.



KICHANG KIM received the B.S. degree in mathematics and the M.S. degree in software convergence from Soongsil University, Seoul, South Korea, in 2017 and 2019, respectively. His research interests include mobile application security, mobile platform security, and machine learning.

JINSUNG KIM received the B.S. degree in computer science and engineering from the Korea National University of Transportation, Chungju, South Korea, in 2018. He is currently pursuing the M.S. degree in software convergence with Soongsil University. His research interests include mobile application security, mobile platform security, and machine learning.

EUNBYEOL KO received the B.S. degree in mathematics from Soongsil University, Seoul, South Korea, in 2019, where she is currently pursuing the M.S. degree in software. Her research interests include mobile application security, mobile platform security, and machine learning.

JEONG HYUN YI received the B.S. and M.S. degrees in computer science from Soongsil University, Seoul, South Korea, in 1993 and 1995, respectively, and the Ph.D. degree in information and computer science from the University of Californian at Irvine, in 2005. He was a Member of Research Staff with the Electronics and Telecommunications Research Institute, South Korea, from 1995 to 2001. From 2000 to 2001, he was a Guest Researcher with the National Institute of Standards and Technology, Gaithersburg, MD, USA. He was a Principal Researcher with the Samsung Advanced Institute of Technology, South Korea, from 2005 to 2008. He is currently an Associate Professor with the School of Software and the Director of the Cyber Security Research Center, Soongsil University. His research interests include mobile security and privacy, the Internet of Things security, and applied cryptography.