Research Article

ACAMA: Deep Learning-Based Detection and Classification of Android Malware Using API-Based Features

Eunbyeol Ko,¹ Jinsung Kim,² Younghoon Ban,² Haehyun Cho ¹ and Jeong Hyun Yi ¹

¹*School of Software, Soongsil University, Seoul 06978, Republic of Korea*

²*School of Software Convergence, Soongsil University, Seoul 06978, Republic of Korea*

Correspondence should be addressed to Haehyun Cho; haehyun@ssu.ac.kr

Received 28 October 2021; Accepted 14 December 2021; Published 29 December 2021

Academic Editor: Ilsun You

Copyright © 2021 Eunbyeol Ko et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

As a great number of IoT and mobile devices are used in our daily lives, the security of mobile devices is being important than ever. If mobile devices which play a key role in connecting devices are exploited by malware to perform malicious behaviors, this can cause serious damage to other devices as well. Hence, a huge research effort has been put forward to prevent such situation. Among them, many studies attempted to detect malware based on APIs used in malware. In general, they showed the high accuracy in detecting malware, but they could not classify malware into detailed categories because their detection mechanisms do not consider the characteristics of each malware category. In this paper, we propose a malware detection and classification approach, named ACAMA, that can detect malware and categorize them with high accuracy. To show the effectiveness of ACAMA, we implement and evaluate it with previously proposed approaches. Our evaluation results demonstrate that ACAMA detects malware with 26% higher accuracy than a previous work. In addition, we show that ACAMA can successfully classify applications that another previous work, AVClass, cannot classify.

1. Introduction

By 2025, it is expected that there will be 55.9 billion connected devices worldwide and 79.4 ZB of data generated by IoT devices [1], and 9 billion smartphones will be connected by 2024 [2]. Accordingly, the use of mobile devices is increasing rapidly. In addition, the mobile application market is also growing. Unfortunately, the attackers exploit the growing ecosystem, and we have observed that the number of mobile malware also increases rapidly [3, 4]. Among the mobile malware, Android malware occupies the largest proportion of mobile malware as Android has the largest share in the mobile application market. An attacker abuses Android's open market policy to inflict damage such as personal information leaks or financial loss to users. Therefore, it is critical to protect users from malware by accurately and quickly detecting Android malicious applications. In addition, to quickly analyze and respond to malicious applications, it is very important to identify their behaviors and classify them.

By identifying or categorizing malicious behaviors, we can help the analysts further analyze characteristics of

malicious applications. On the other hand, identifying malware based on its behavior can let the security analysts to put their efforts on more malware. However, to our knowledge, antivirus products cannot detect unknown malicious applications, and, thus, many studies were conducted to detect unknown malware.

Recently, many studies used Application Programming Interfaces (APIs) as features to detect malware [5–9]. Malicious behaviors must be implemented with a series of specific APIs, and, thus, previously proposed systems which analyze APIs could detect malware with the high accuracy. However, previous APIs-based malware detection systems cannot classify malware into detailed categories. Therefore, for developing malware classification techniques, the research community had to conduct other studies. For example, AVClass [10] uses antivirus vendors' reports obtained from VirusTotal. In general, antivirus vendors do not have a common analysis result of malware. Consequently, the results of AVClass are not reliable.

In this work, we propose ACAMA that can identify malware and can classify malware into specific categories by

utilizing APIs used to implement malicious functions. ACAMA generates deep learning models based on APIs of Android malware with the CNN algorithm. To evaluate the performance of ACAMA, we compare it with the performance of a previous approach proposed by Kim et al. [11]. We, also, evaluate the effectiveness of ACAMA by using the classification results of AVClass. Overall, the evaluation results show that we used the same feature that Kim et al. used, but ACAMA detects malware with the higher degree of precision. ACAMA detects malware with 95% accuracy, 26% higher than Kim et al. In addition, ACAMA could successfully classify 72.456% of malware that AVClass could not classify.

In summary, this paper makes the following contributions:

- (i) We propose a novel malware detection and classification approach, named ACAMA, that uses APIs-based features. To develop this, we learn APIs extracted from our dataset using the CNN deep learning algorithm.
- (ii) We develop the predicted model of ACAMA and evaluate it. Also, to show the effectiveness of ACAMA, we compare the evaluation results with the results of Kim et al. and AVClass. The evaluation result shows that ACAMA detects malware with much higher accuracy than Kim et al. It also shows that ACAMA can classify malware that AVClass cannot categorize.

2. Background and Related Work

In this section, we introduce categories of mobile malware and those behaviors. In addition, we discuss previous approaches for detecting malware by using APIs as a feature and other related work.

2.1. API-Based Approaches. Many approaches were proposed to analyze malicious applications using their APIs [5–9, 11–17]. DroidAPIMiner [6] detected malware with k -Nearest Neighbors (k -NN) [18] by using APIs in malware. DroidDelver [8] extracted API calls blocks from the small files. By using generated API call blocks, they employed Deep Belief Network (DBN) for Android malware detection. MalDozer [17] tried to detect malware by using the CNN based on API calls, extracted from DEX assembly. Nix et al. [9] also detected malware from API calls in applications by using the CNN. However, most of the previously proposed approaches focused on the binary classification problem (i.e., identifying malware from benign applications). In this section, we discuss two systems closely related to our work. AVIS [5] ensembles 10 types of machine learning algorithms, such as Support Vector Machine [19], Naive Bayes [20], and k -NN, and directly score the API to create the API ranking. In addition to detecting malicious applications, AVIS evaluates an application quantitatively through the average value to provide a risk indicator. However, in machine learning, you need to use appropriate algorithms according to the data to obtain accurate results. However,

not all of the algorithms used can be considered suitable for API data. Also, the method of bagging ensemble other algorithms is not suitable for classifying malware into specific categories [5, 11].

Kim et al.'s [11] methods, like AVIS, directly scored APIs to create API rankings and quantitatively evaluated applications. However, unlike AVIS, the application was quantitatively evaluated using a weighted average value. In addition, since the bagging ensemble was performed using only XGBoost [21], the accuracy was improved compared to the previous study. However, the XGBoost algorithm itself uses the boosting technique, which is already an ensemble technique. Furthermore, since the bagging technique is also used, the cost of classifying applications is high. In addition, there may be a problem of objectivity, such as overfitting, because a person used the feature after specifying the range through an experiment when selecting the training data.

2.2. Categories of Mobile Malware. Analyzing malware is an error-prone task. Therefore, if a category of malware can be determined automatically, it can provide boundaries of analyses to the analysts to help them with conducting effective and efficient analysis. However, it is challenging to automatically classify malware. Moreover, antivirus systems use different malware categories, and even the same applications can be categorized differently by each antivirus system. Among the previous studies, Samra et al. [22] extracted permissions from the manifest files and classified only two categories: business and tools with the k -means algorithm. On the other hand, DroidMiner [23] proposes a two-level behavioral graph model and extracts sensitive execution paths from Android program logic. They classified malware into 12 families by using the Random forest.

In this work, we classify malware based on its behavior inferred by APIs used to implement the malware, which would help security analysts by providing an instinctive comprehension of malware behavior. Since there are many categories of malware, we use categories of malware proposed by Wei et al. [24]. Wei et al. used 24,560 malicious applications to classify them into 71 categories. Table 1 shows the categories proposed by Wei et al.

2.3. Labeling Android Malware. A typical technique to analyze malicious applications and categorize them is to use VirusTotal [25]. From VirusTotal, users can obtain antivirus scanning reports for an application and can categorize the application using words contained in the report.

EUPHONY [26] analyses all labels provided by different vendors for labeling malware families. Then, it builds a graph representing the association links between family names based on labels that they assigned to the malware samples. Finally, EUPHONY used Prim's algorithm to transform the graph into a Minimum Spanning Tree. Therefore, EUPHONY unifies malware labels into common family groups, while Li et al. [27] remove legitimate library code from applications for labeling them. Then, it used the malicious payload mining method with 68 malware labels to cluster malware.

TABLE 1: Categories of Android malware proposed by Wei et al. [24].

Category	Type	Category	Type
Airpush	Adware	Mecor	Trojan-Spy
AndroRAT	Backdoor	Minimob	Adware
Andup	Adware	Mmarketpay	Trojan
Aples	Ransom	MobileTX	Trojan
BankBot	Trojan-Banker	Mseg	Trojan
Bankun	Trojan-Banker	Mtk	Trojan
Boqx	Trojan-Dropper	Nandrobox	Trojan
Boxer	Trojan-SMS	Obad	Backdoor
Cova	Trojan-SMS	Ogel	Trojan-SMS
Dowgin	Adware	Opfake	Trojan-SMS
DroidKungFu	Backdoor	Penetho	HackerTool
Erop	Trojan-SMS	Ramnit	Trojan-Dropper
FakeAV	Trojan	Roop	Ransom
FakeAngry	Backdoor	RuMMS	Trojan-SMS
FakeDoc	Trojan	SimpleLocker	Ransom
FakeInst	Trojan-SMS	SlemBunk	Trojan-Banker
FakePlayer	Trojan-SMS	SmsKey	Trojan-SMS
FakeTimer	Trojan	SmsZombie	Trojan-Spy
FakeUpdates	Trojan	Spambot	Backdoor
Finspy	Trojan-Spy	SpyBubble	Trojan-SMS
Fjcon	Backdoor	Stealer	Trojan-SMS
Fobus	Backdoor	Steek	Trojan-Clicker
Fusob	Ransom	Svpeng	Trojan-Banker
GingerMaster	Backdoor	Tesbo	Trojan-SMS
GoldDream	Backdoor	Triada	Backdoor
Gorpo	Trojan-Dropper	Univert	Backdoor
Gumen	Trojan-SMS	UpdtKiller	Trojan
Jisut	Ransom	Utchi	Adware
Kemoge	Trojan-Dropper	Vidro	Trojan-SMS
Koler	Ransom	VikingHorde	Trojan-Dropper
Ksapp	Trojan	Vmvol	Trojan-Spy
Kuguo	Adware	Winge	Trojan-Clicker
Kyview	Adware	Youmi	Adware
Leech	Trojan-SMS	Zitmo	Trojan-Banker
Lnk	Trojan	Ztorg	Trojan-Dropper
Lotoor	HackerTool	Total 71 categories	Total 24,650 apps

AVClass [10] is a malware labeling system based on VirusTotal reports. AVClass does not simply count specific words of the AV scanning reports to determine a category, it creates aliases so that similar categories can be grouped into the same category through word learning in advance. Then, it compares AV reports and alias of an unknown application, arranging words of the reports. Finally, words that appear twice or more in the same category are set as an application category. AVClass is generally more accurate than specifying categories using simple word counting, but the accuracy is still not high enough because the limited number of words used to classify malware. Also, AVClass strongly relies on AV reports, and, thus, if antivirus systems cannot generate

enough information regarding a malicious application, AVClass cannot categorize them.

2.4. Deep Learning Visualization for Interpretation of Classified Result. Deep learning has the advantage that automatically uses feature engineering. It is, thus, easy to learn a predictive model using deep learning algorithms. However, it is difficult to know the basis of which features are used to learn the predictive model. Therefore, to justify prediction results, we need to interpret them while the predictive model is processing data.

In this work, we provide the confidence level in the classification results using LIME [28]. LIME is model-agnostic and, thus, can be used with CNN, LSTM, decision tree, and the other machine learning algorithms. Even if we replace the underlying machine learning algorithm, we can still use the same visualization model for the interpretation. Furthermore, LIME is an algorithm that can explain the predictions of any classifier in a precise way, by approximating it locally with an interpretable model. LIME inserts slightly changed input values into the predictive model and considers the input value with a significant change in the predicted value as an important value. We use this value as the interpretation of the prediction results.

3. Design

In this section, we describe our goal and details of the proposed approach.

3.1. Goal. The main goal of this work is not only detecting malware but also classifying them into specific categories using deep learning. Classifying malware can quickly detect the attack of malware due to selecting and analyzing representative applications for each family. Therefore, analysts can protect users from attacks of malware. Also, based on characteristics of malware that can be deterministically obtained, we aim to avoid misclassifying malware rather than relying on reports generated by antivirus vendors.

3.2. Overview of ACAMA. In order to achieve the goal, we design and implement a deep learning-based approach, named ACAMA, that use APIs of an application as a feature. ACAMA mainly consists of three stages: Preprocessor, Deep Learner, and Categorizer. In the first stage, the Preprocessor extracts APIs from labelled benign and malicious applications using AndroGuard [29] and generates training datasets. In the second Deep Learner stage, the API is vectorized, and then CNN learns the vectorized dataset. After that, CNN creates a classifier model. In the last Categorizer stage, unknown applications are entered into the classifier model created in the previous step for classifying malicious applications. At that time, when the classifier detects a malicious application, it is transmitted to the category classifier. After the category classifier categorizes the malware, ACAMA using LIME provides a report that shows important APIs used to classify the

malware to users. The overall structure of the proposed method is shown in Figure 1.

3.3. Preprocessor for Training Dataset. ACAMA extracts APIs to generate the training dataset using AndroGuard. API extraction process is performed by parsing the classes.dex file containing the actual code of an application. Method Table and Class Def Table among the eight DEX file fields contain method-related data. Method Table contains user-defined methods and framework APIs, which are all method names used in the application. Class including framework API is not defined in DEX file but implemented in Android device. Therefore, Class Def Table contains an only user-defined class list. In Class Data Item in Class Def Table, Direct Methods and Virtual Methods contain method information of each class. In particular, the offset of the method points to the fact that the index of the method table is included in the direct method. If you go to the String Table using the Name index value of the Method Table, you can find the API in the Data Section and extract it. Table 2 shows the characteristic information of an API that can be obtained by extracting it in this way. Since the dimension can be too large due to the number of words, the training set is constructed using only the method name that can express the API as much as possible, excluding the description. We generate two types of training sets: the first dataset, which collects 10,000 benign applications and 10,000 malicious applications, and the second dataset, which collects only malicious applications for which categories are determined. we use the Android Malware Dataset (AMD) [24] for collecting malware.

3.4. Learning the Training Datasets Using CNN. When using CNN algorithm, the feature map is extracted through convolution operations by applying a filter to the data value. Therefore, we need to vectorize (word embedding) the API features that we extracted. Methods for vectorizing natural language include One-Hot, Word2-Vec, Glove, BOW, TF-IDF, and Tokenize. Among them, we use the tokenize method that assigns numbers to words by mapping integers by making a dictionary of words existing in the API feature. In our dataset, a dictionary consisting of a total of 1,273,251 words is used by adding two additional methods including padding to match the size of the vector and cover unknown APIs that are not used in the training phase. If we simply map a number, the dimension is too large and the relationship between the APIs cannot be considered, so an embedding layer is used to adjust the vector value. To input to the embedding layer, the size of each application's API vector is adjusted by padding it. After that, ACAMA learns using the embedding layer to transform the vector into a dense vector that can contain a lot of information compared to a small dimension. In this paper, the size of the dense vector is set to 64. Figure 2 shows an example of the vectorization process.

3.4.1. CNN Parameter Locations for Optimization. After we determined the data format, we optimized the parameters of the CNN algorithm in order to learn datasets effectively. If appropriate parameters are not used based on the characteristics of datasets, the performance of a classifier will be low. Therefore, the parameters should be optimized through repeated experiments. Specifically, in this work, we optimized the embedding dimension, the number and size of filters, the type and size of pooling, and the number of convolutional layers. Figure 3 shows the location of each parameter.

3.4.2. Learning Datasets with CNN. Since the ACAMA uses API features, the convolution operation is performed using the Conv1D layer, which is widely used in Text-CNN. The dimension used in the operation was set to 8 (embedding dimension is 8). Therefore, the size of the filter was (64, 8), and the number of filters was set to 32. Also, we used padding and stride. As default values, we set padding to be 0 and stride to be 1. Hence, the size of the feature map becomes (7765, 32) because the filter length is set to 8 and the number of filters is set to 32. Since the proposed method uses two datasets, learning phase is also performed twice with each dataset. Figure 4 shows the data and filters during the learning process of the category classifier.

After that we apply an activation function to the feature map created in the previous step. In ACAMA, we used the ReLU function [30] as an activation function. The ReLU function is one of the most widely used activation functions because it can learn relatively quickly and the computation cost is not high. After that, using the max-pooling on the generated activation map, the largest vector is selected from the feature vectors. This process allows us to pick the most prominent feature used in an application. We set the pooling size to be 1.

Lastly, through the process of making the result value from the pooling layer into a one-dimensional vector, we used the softmax function that normalizes all outputs to values between 0 and 1. Based on this output, the risk classifier detects malicious applications and the category classifier classify them.

3.5. Malware Detection and Categorization. The classifier created using the training dataset containing both benign and malicious application is called a risk classifier, which classifies whether an application is a benign application or a malicious application. And if it is a malicious application, the application is transferred to a *category classifier* created using the dataset of labelled malicious applications. The category classifier outputs a probability vector for each category of the received unknown application.

3.5.1. Identifying Categories of Malware. Since there are only 71 categories, the probability results can be ambiguous for malicious applications that require more detailed categories. Therefore, in this work, a malicious application that does not have a classification result higher than 0.5 for all the

FIGURE 1: Architecture of ACAMA.

TABLE 2: Characteristic information of sendDataMessage API.

Package name	Android.telephony
Class name	SmsManager
Method name	SendDataMessage
Description	Send a data-based SMS to a specific application port.

categories is called “Unlabelled applications.” For such applications, ACAMA provides a list of categories where the probability is close to 0.5 with main features (APIs) of the application to users so that, at least, users can understand possible behaviors of malware.

To this end, using the LIME algorithm, ACAMA provides reasons of classification results of the classifier. First, LIME converts the input features to find important APIs and creates several similar input values. Then, important features are determined based on the classification result that comes out by putting the converted similar input value into the category classifier. In this work, 10 important APIs are used to provide a confidence indicator for the results, and the goal is to be able to know what kind of actions are possible by a malicious application, even if there is no label (the application cannot be categorized with the high confidence). Figure 5 shows the process of LIME that extracts important APIs from an application.

4. Evaluation

In this section, we evaluate the proof-of-concept implementation of ACAMA.

4.1. Experimental Setup. We used 2 Intel® Xeon® Gold 6230 20-Core 2.10 GHz CPUs, 128 GB RAM, 4 NVIDIA GeForce RTX 2080 Ti GPUs, and Ubuntu 18.04.4 LTS. Also, with

Python 3.7.6 version, AndroGuard 3.3.5, Keras 2.2.4, and TensorFlow 1.14.0 were used to implement ACAMA.

In addition, we used 10,000 benign applications randomly selected from the Google Play Store [31] for detecting malware, and 10,000 malicious applications from the AMD [24] were used as the training dataset. Also, we used 10,000 malicious applications collected from VirusShare [32] for evaluating the performance of ACAMA.

If the data is too biased, it can interfere with learning; 20,000 out of 24,090 of the AMD were used, and 71 categories specified in the dataset were used as well. Moreover, as the category classification dataset, 10,133 applications that AVClass cannot classify were used to evaluate ACAMA’s effectiveness of classifying malicious applications.

4.2. Parameter Setup. In this paper, we set the parameters that maximize the performance of the CNN classifier based on the loss rate and accuracy by using the training datasets as validation data. By changing from the most basic structure to the most commonly used parameter values, we found well-optimized parameters including embedding dimension, number of filters, filter size, type and size of pooling, and number of convolution layers. The CNN parameters used in the proposed method are as shown in Table 3.

4.2.1. Embedding Dimension. Commonly used embedding dimensions are 50, 64, 100, 150, and 200. The results of the experiment are shown in Figure 6. Since the accuracy of the risk classifier gradually decreases from 100 dimensions and the loss rate increases, we decided that there was no need to experiment further by increasing the dimension; we tested up to 200 dimensions. The category classifier showed the highest accuracy and lowest loss rate when we use 64 dimensions. As in the two graphs of Figure 6, both the risk

FIGURE 2: The process of transform the APIs to vectors.

FIGURE 3: Location of parameters of CNN.

FIGURE 4: Process of data and filter convolution.

classifier and the category classifier showed the lowest loss rate and the highest accuracy when the embedding dimension was 64, and, thus, we determined the API embedding dimension to be 64.

4.2.2. The Number of Filters. In text-CNN, the number of filters is usually specified as a related number such as a factor or multiple of the embedding dimension. Since the embedding dimension is 64, we experimented with 16 and 32, and 100, 150, 200, 250, and 300 to find the approximate range. The experimental results are shown in Figure 7. As the graphs show, we obtained the best results when we used 32 filters. As the number of filters increases, the number of parameters increases, and the efficiency decreases. Therefore, the number of filters was determined to be 32 for both classifiers based on the above results.

4.2.3. Size of Filter. Once the number of filters was determined, we have to determine the size of the filters. Usually,

using a smaller filter than a larger filter reduces the number of parameters and has better performance. As shown in Figure 8, the experimental results demonstrate that, in both classifiers, we obtained the best results when the filter size is 8. Hence, we determined the size of filters to be 8 according to the evaluation results.

4.2.4. Pooling and the Number of Convolution Layers. It is a well-known fact that the performance of text-CNN is better when the max-pooling is used than the average pooling is used [33]. In addition, in the case of text-CNN, the max-pooling size 1 showed the best performance [34]. Therefore, ACAMA also used the max pooling and set the pooling size to be 1.

In addition, the number of layers of the convolution layer is also an important parameter, but since the data and the proposed method in this paper have already obtained high verification accuracy and low verification loss rate with one convolution operation, it is necessary to increase the

FIGURE 5: Process of LIME for extracting important APIs from an application.

TABLE 3: ACAMA used parameters of CNN.

Embedding dimension	64
The number of filters	32
Size of filter	8
Pooling	Max-pooling
Size of pooling	1
The number of convolutional layers	1

(a)

(b)

FIGURE 6: Experimental results of embedding dimensions of risk classifier (a) and category classifier (b).

FIGURE 7: Experimental results for the number of filters in the risk classifier (a) and category classifier (b).

FIGURE 8: Filter size experiment results for risk classifier (a) and category classifier (b).

layer to make the calculation more complicated, which was judged not to be necessary. Finally, the number of convolutional layers is set to be 1.

4.3. Evaluation Results. The results of detection of malicious applications and classification results of malicious application categories are described.

4.3.1. Malware Detection Result. Figure 9 shows the accuracy comparison results between ACAMA and a malicious application detection system proposed by Kim et al. [11], closely related with ACAMA, that use APIs as a feature. The verification accuracy is a result of malicious detection by reapplying the list to the training dataset after removing 10% of it to verify the effectiveness of the classifier. On the other hand, the test data accuracy refers to the accuracy when we

used a new dataset that is not contained in the training dataset.

In the training verification data, we can observe that the accuracy of ACAMA is similar to the one of Kim et al.'s approach. However, ACAMA outperforms the previous approach when we used a new dataset.

4.3.2. Category Classification Result. Table 4 shows the classification results of AVClass using malicious applications in VirusShare. The undetected column of Table 4 indicates the number of applications that VirusTotal did not detect as malicious applications. The unlabelled column indicates the number of applications detected as malware by VirusTotal but could not be classified by AVClass.

However, ACAMA classified a total of 10,133 malicious applications that AVClass could not classify, and the results are

FIGURE 9: Accuracy of detection result of ACAMA and Kim et al.

TABLE 4: AVClass [10] experimental results.

	Labelled	Undetected	Unlabelled	Undetected + unlabelled
2016	9,978	495	2,380	2,875
2017	7,174	984	1,994	2,978
2018	24,352	1,069	3,211	4,280
Total	41,504	2,548	7,585	10,133

TABLE 5: Category classification result of ACAMA.

	Not extracted	Labelled	Unlabelled
The number of malware	1,165	7,342	1,626

TABLE 6: Detailed classification results of ACAMA.

	Airpush	Dowgin	Gumen	Minimob	MobileTX	Tesbo
The number of malware	4,109	577	1,659	101	784	112

TABLE 7: Result of important API output.

Important APIs (Gumen)
Lcom/umeng/analytics/AdsConnectTask;
Lcom/Google/analytics/tracking/android/AdMobInfo;
Landroid/animation/AnimatorSet; playSequentially;
Landroid/support/v7/app/AppCompatActivity; setContentView;
Landroid/support/v4/widget/CursorAdapter; bindView;
Lorg/apache/http/Header; getElements;
Landroid/support/v4/text/ICUCompat; getScript;
Landroid/widget/PopupMenu/getDragToOpenListener;
Ljava/lang/IllegalArgumentException;
Landroid/view/View; getRotation;

shown in Table 5. Consequently, we can classify 7,342 applications out of 10,133 ones that AVClass could not classify. ACAMA shows the detailed classification results in Table 6.

In addition, we checked the classification results by using LIME. To this end, we extracted main features of each malware and manually verified 7,342 classification results to check whether the classification results are correct or not. As a result, by verifying evaluation results with LIME, we found that our category classifier can classify malicious applications without a misclassified result. Table 7 shows important APIs that LIME found, which is used to implement malicious functions of malware of the Gumen category which behaves similar to the Trojan-SMS malware family.

5. Conclusions

In this paper, we proposed ACAMA that identifies malware and classifies malware into specific categories based on behavioral characteristics of malware. We evaluated ACAMA by comparing its performance with a previous approach proposed by Kim et al. [11]. We also evaluated the effectiveness of ACAMA with AVClass. In summary, the evaluation results show that ACAMA outperforms the previous approach proposed by Kim et al. [11]. Also, we observed that ACAMA can classify 72.456% of malware that AVClass cannot classify.

However, ACAMA needs well-labelled dataset to categorize malware due to the use of supervised learning. Also, since ACAMA only uses the Android framework API, we cannot avoid the out-of-vocabulary problem (i.e., if malware is obfuscated or malware uses APIs that ACAMA did not catch in the learning phase, it cannot classify the malware). We leave these limitations as future work.

Data Availability

The data used to support the findings of this study were supplied by Eunbyeol Ko under license and so cannot be made freely available. Requests for access to these data should be made to Eunbyeol Ko (kongstar159@soongsil.ac.kr).

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This work was supported by the Institute for Information & Communication Technology Planning & Evaluation (IITP) grant funded by the Korean Government Ministry of Science and ICT (MSIT) (no. 2017-0-00168, Automatic Deep Malware Analysis Technology for Cyber Threat Intelligence).

References

- [1] *IoT Growth Demands Rethink of Long-Term Storage Strategies*, Says IDC. IDC, Massachusetts MA USA, 2020.
- [2] *Ericsson Mobility Report*, ERICSSON, Stockholm, Sweden, 2020.
- [3] *McAfee Labs Threats Report*, McAfee, San Jose, CA, USA, 2020.
- [4] *Mobile Malware Report-No Let-Up with Android Malware*, GDATA, Bochum, Germany, 2019.
- [5] H. Kim, T. Cho, G.-J. Ahn, and J. Hyun Yi, "Risk assessment of mobile applications based on machine learned malware dataset," *Multimedia Tools and Applications*, vol. 77, no. 4, pp. 5027–5042, 2018.
- [6] Y. Aafer, W. Du, and H. Yin, "DroidAPIMiner: mining API-level features for robust malware detection in android," in *Proceedings of the International Conference on Security and Privacy in Communication Systems*, pp. 86–103, Springer, Kansas KS USA, September 2013.
- [7] A. Desnos and G. Gueguen, "Android: From Reversing to Decompilation," in *Proceedings of the Black Hat Abu Dhabi*, pp. 77–101, 2011.
- [8] S. Hou, A. Saas, Y. Ye, and L. Chen, "DroidDelver: an android malware detection system using deep Belief Network based on API call blocks," in *Proceedings of the International Conference on Web-Age Information Management*, pp. 54–66, Springer, Lagos, Nigeria, June 2016.
- [9] R. Nix and J. Zhang, "Classification of android apps and malware using deep neural networks," in *Proceedings of the 2017 International Joint Conference on Neural Networks (IJCNN)*, pp. 1871–1878, IEEE, Anchorage, AK, USA, May 2017.
- [10] M. Sebastián, R. Rivera, P. Kotzias, J. Caballero, and AVClass, "A tool for massive malware labeling," in *Proceedings of the International Symposium on Research in Attacks, Intrusions, and Defenses*, pp. 230–253, Springer, Paris, France, September 2016.
- [11] K. Kim, J. Kim, E. Ko, and J. H. Yi, "Risk assessment scheme for mobile applications based on tree boosting," *IEEE Access*, vol. 8, Article ID 48503, 2020.
- [12] K. Kim, E. Ko, J. Kim, and J. H. Yi, "Intelligent malware detection based on hybrid learning of api and acg on android," *Journal of Internet Services and Information Security*, vol. 9, no. 4, pp. 39–48, 2019.
- [13] J. Jung, H. J. Kim, S. je Cho, S. Han, and K. 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, 2019.
- [14] I. K. Cho, T. Kim, Y. J. Shim, H. Park, B. Choi, and E. G. Im, "Malware similarity analysis using api sequence alignments," *Journal of Internet Services and Information Security*, vol. 4, no. 4, pp. 103–114, 2014.
- [15] M. Park, G. You, S. je Cho, M. Park, and S. Han, "A framework for identifying obfuscation techniques applied to android apps using machine learning," *Journal of Wireless Mobile Networks Ubiquitous Computer Dependable Application*, vol. 10, no. 4, pp. 22–30, 2019.
- [16] M. Park, J. Seo, J. Han, H. Oh, and K. Lee, "Situational awareness framework for Threat intelligence measurement of android malware," *Journal of Wireless Mobile Networks Ubiquitous Computer Dependable Application*, vol. 9, no. 3, pp. 25–38, 2018.
- [17] E. B. Karbab, M. Debbabi, A. Derhab, D. Mouheb, and MalDozer, "MalDozer: Automatic framework for android malware detection using deep learning," *Digital Investigation*, vol. 24, pp. S48–S59, 2018.
- [18] T. Cover and P. Hart, "Nearest neighbor pattern classification," *IEEE Transactions on Information Theory*, vol. 13, no. 1, pp. 21–27, 1967.
- [19] C. Cortes and V. Vapnik, "Support-vector networks," *Machine Learning*, vol. 20, no. 3, pp. 273–297, 1995.
- [20] E. Frank and R. R. Bouckaert, "Naive bayes for text classification with unbalanced classes," in *Proceedings of the European Conference On Principles Of Data Mining And Knowledge Discovery*, pp. 503–510, Springer, Berlin, Germany, September 2006.
- [21] T. Chen and C. Guestrin, "Xgboost: a scalable tree boosting system," in *Proceedings of the 22nd Acm Sigkdd International Conference on Knowledge Discovery and Data Mining*, pp. 785–794, San Francisco, CA, USA, August 2016.
- [22] A. A. A. Samra, K. Yim, and O. A. Ghanem, "Analysis of clustering technique in android malware detection," in *Proceedings of the 2013 Seventh International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing*, pp. 729–733, Taichung, Taiwan, July 2013.
- [23] C. Yang, Z. Xu, G. Gu, V. Yegneswaran, and P. Porras, "DroidMiner: automated mining and characterization of fine-grained malicious behaviors in android applications," in *Proceedings of the European Symposium Computer Security - ESORICS 2014*, pp. 163–182, Wroclaw, Poland, September 2014.
- [24] F. Wei, Y. Li, S. Roy, X. Ou, and W. Zhou, "Deep ground truth analysis of current android malware," in *Proceedings of the International Conference On Detection Of Intrusions And Malware, and Vulnerability Assessment*, pp. 252–276, Springer, Bonn, Germany, July 2017.
- [25] Virustotal, <https://www.virustotal.com/>, 2020.
- [26] M. Hurier, G. T. Suarez, S. K. Dash et al., "Euphony: Harmonious unification of cacophonous anti-virus vendor labels for android malware," in *Proceedings of the 2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*, pp. 425–435, Buenos Aires, Argentina, May 2017.
- [27] Y. Li, J. Jang, X. Hu, and X. Ou, "Android malware clustering through malicious payload mining," in *Proceedings of the International Symposium on Research in Attacks, Intrusions, and Defenses*, pp. 192–214, San Sebastian, October 2017.
- [28] M. T. Ribeiro, S. Singh, and C. Guestrin, "Why should i trust you? Explaining the predictions of any classifier," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 1135–1144, San Francisco, CA, USA, August 2016.
- [29] "Androguard," 2020, <https://github.com/androguard/androguard>.
- [30] V. Nair and G. E. Hinton, "Rectified linear units improve restricted Boltzmann machines," in *Proceedings of the 27th*

- International Conference on Machine Learning (ICML-10)*, Haifa, Israel, June 2010.
- [31] “Google play store,” 2019, <https://play.google.com/store/%20apps>.
 - [32] “VirusShare,” 2020, <https://www.virusshare.com/>.
 - [33] A. Conneau, H. Schwenk, L. Barrault, and Y. Lecun, “Very Deep Convolutional Networks for Text Classification,” 2016, <https://arxiv.org/abs/1606.01781>.
 - [34] Y. Zhang and B. C. Wallace, “A sensitivity analysis of (and practitioners’ guide to) convolutional neural networks for sentence classification,” in *Proceedings of the Eighth International Joint Conference on Natural Language Processing*, vol. 1, pp. 253–263, Taipei, Taiwan, November 2017.