Enhanced Android Malware Detection: An SVM-based Machine Learning Approach

Hyoil Han
School of Information Technology
Illinois State University
Normal, IL, USA
hhan12@ilstu.edu

SeungJin Lim
Department of Computer Science
Merrimack College
North Andover, MA, USA
lims@merrimack.edu

Kyoungwon Suh
School of Information Technology
Illinois State University
Normal, IL, USA
kwsuh@ilstu.edu

Seonghyun Park
Dept. of Applied Computer Engineering
Dankook University
Yongin, Republic of Korea
qkrtjdgus268@dankook.ac.kr

Seong-je Cho
Dept. of Computer Science and Engineering
Dankook University
Yongin, Republic of Korea
sjcho@dankook.ac.kr

Minkyu Park
Dept. of Software Technology
Konkuk University
Chungju, Republic of Korea
minkyup@kku.ac.kr

***Abstract*—The cybersecurity of increasing numbers of mobile devices and their users are threatened by malicious applications. Detecting malicious Android applications is a challenge due to the massive number of Android applications and their various properties which provide a large set of features and a sparse dataset. We focus on the resources the Android applications call and employ the Application Program Interface (API) calls as features. The dataset used in this work is from an Android environment where malicious and benign applications frequently access the system resources through Android API calls. Since an Android application would invoke a relatively small number of APIs in ordinary scenarios, data in the dataset is inherently sparse and high dimensional. We experimented intensively with 58,602 Android applications as well as 133,227 features (i.e., API Calls). This paper presents a machine-learning-based approach using Support Vector Machines (SVM) to detect malicious Android applications; the new approach delivers results highly competitive with existing approaches.**

***Index Terms*—Android Malware Detection, API Calls, Support Vector Machines, Machine Learning.**

I. Introduction

We live in an era in which mobile devices are widely used. More than 2.5 billion out of 7.7 billion people in the world are smartphone users and approximately 5.1 billion people use mobile devices in the world today [1]. Therefore, more than 65% of people are in the vulnerable situation while they use mobile apps. Every mobile device (smartphones and tablets) has a lot of mobile applications (called apps). People use smartphones/tablets for their banking, VoIP calls and their access to important web sites that require their login/password. Malicious apps can be dangerous to smartphone/tablet users because such apps can threaten the cybersecurity and privacy of smartphone/tablet users. Detecting malicious apps can help users be more secure in using smartphones/tablets and protect their data as well as privacy.

We propose a machine-learning-based malware detection approach and show how our approach presents competitive results compared to existing approaches. We are especially interested in and use Android apps in this paper because Android is the most popular mobile operating system and has a huge number of malware attacks. Android occupies more than 70% of the world mobile operating system market [2]. Detecting malicious Android applications will greatly help improve cybersecurity overall. However, other mobile operating systems also can use our approach to detect malicious apps. Android applications (i.e., apps) call many APIs. We treat the API calls as features. Each API call with its specific argument and return type is considered a distinct feature. Each feature is represented by a triple (*API method name, argument, return type*) invoked by an Android application. The same API calls are considered different features when they have different arguments and/or return types. Therefore, each app in our dataset has many features and our dataset becomes high-dimensional. More specifically, our dataset has a vast number of variables (or features). In our experiment, the data for any given app is sparse because each app uses relatively few out of the vast number of features.

We classify the Android App data using Support Vector Machines (SVM). SVM is a statistical learning algorithm that uses kernel methods [3]. SVM is used widely in many applications such as text categorization [4], [5], image identification [6], [7], and bioinformatics [8], [9]. However, SVM is used somewhat less in malware detection. Malware datasets have a high dimension and the data is sparse. These characteristics present many challenges in classifying the data. Our work shows that such sparse and high-dimensional data can be well classified using SVM, and our approach achieves more than 99% classification accuracy.

The dataset we used is from the work in [10], [11] as well as from [12], [13]. The dataset for our malware detection has two groups: one for benign apps and the other for malicious apps. The benign dataset used in our experiment includes the 28,489 Android apps collected from Google Play, Amazon AppStore and APKPure (https://apkpure.com/) during December 2016-

February 2017 [10], [11]. On the other hand, the malicious dataset used in our work is from AMD [13] and Drebin [12] datasets in which the total number of malware apps is 30,113. The data has 133,227 attributes and each of them represents an invoked API call as well as its arguments and return type. In our work, each API call is identified by its API method name, argument, and return type. We disregard the relationships between attributes (i.e., API calls) and treat each attribute (i.e. API call) independently. Each app is treated as an instance in our work. For malware detection, other researchers use sequences of API calls [14], semantic relationships of APIs between API calls [15], and communications among components of APIs [16]. Our work exclusively focuses on the static analysis of resources accessed by Android applications, but still produces a very competitive accuracy in the malware detection of Android apps. We consider a data instance is noise if it is associated with no API calls. In our dataset, 5.0% or 1,503 instances out of 30,113 malicious instances were noise while there was no noise in the benign instances. In principle, noise should be removed from the training set in order to build a model from truthful information.

The main contribution of our work is summarized as follows: (1) to achieve highly competitive accuracy and recall metrics in a sparse and high dimensional dataset and (2) to remove noise and decrease the curse of dimensionality.

The paper is organized as follows. Section II reviews related studies and section III introduces and explains our approach. Section IV presents model performance and results. Finally, the last section concludes the paper.

II. Related Work

Malware can be analyzed by static or dynamic techniques. In general, static analysis is computationally more efficient than the existing execution-based dynamic analysis. The approaches such as code obfuscation and packer-based encryption [17], [18] are used to avoid static analysis techniques.

The malware detection approach by Peiravian and Zhu in [19] employed a feature-based learning framework, where API calls and permissions are used as features. The work utilizes machine learning approaches such as SVM, decision trees, and bagging approaches and accomplished its accuracy up to 96.88%. The work in [19] used a static analysis technique and validated that the combination of API calls with permissions provides a good set of features to detect malicious Android apps.

Jung et al. [10] used a static analysis technique and presented the approach to select a set of API calls for detecting malicious Android apps. The chosen sets of API calls are used as classification features. The work selects two set of API calls: a benign API list and a malicious API list, which include the most commonly invoked APIs in benign apps and malicious apps, respectively. To determine if a new app is benign or malicious, Jung et al. [10] calculated the summation of inverse rank of each API used by the app. This is calculated against both the benign API list and the malicious API list. Then they compared the two values: one from the summation of inverse rank with benign API list and the other from malicious API list. The larger value determines the property of the given app, i.e., benign or malicious. This is a very simple approach, but gives a high accuracy near 90%.

For detecting Android malware, Faruki et al. [20] proposed AndroSimilar, a robust method that generates signatures of malware by extracting statistically improbable features. The generated signatures are simply used to detect malware. The method selects the features using similarity digest hashing mechanism and is effective to detect obfuscated or repackaged Android malware. Signature generated with the proposed approach is robust to detect unknown samples transformed by various obfuscation techniques that are not detected by the well-known anti-malware products. AndroSimilar is a syntactic foot-printing mechanism that finds regions of statistical similarity with known malware to detect unknown new samples. It considers syntactic similarity of whole file instead of just opcodes for faster detection compared to known fuzzy hashing approaches. Androsimilar malware signatures database shows 40% similarity with official market apps, which still consists of many less malware signatures compared to commercial anti-malware solutions.

In order to detect obfuscated Android malware, Suarez-Tangil et al. [17] developed DroidSieve, a system for classifying Android malware based on static analysis that is fast, accurate, and resilient to code obfuscation. DroidSieve depends on several features known to be characteristic of Android malware, including API calls, permissions, code structure, and the set of invoked components. It also deeply inspects Android apps to exploit obfuscation-invariant features including native components, embedded assets, and a certificate, which have been missed by existing malware analysis techniques. Suarez-Tangil et al. analyzed the relative importance of their features and showed that artifacts introduced by state-of-the-art obfuscation mechanisms could provide high-quality features for reliable detection and family classification. Using those features, DroidSieve first decides whether a given app is malicious and, if so, classifies it as belonging to a family of related malware. In DroidSieve, malware detection and family identification was implemented using Extra Trees. The robustness of DroidSieve was evaluated on a set of over 100K benign and malicious Android apps. It achieved up to 99.82% accuracy with zero false positives. The same features allowed malware family identification with an accuracy of 99.26%.

On the other hand, Demontis et al. [18] pointed out that machine learning exhibited inherent vulnerabilities that could be exploited to avoid detection at test time. In other words, machine learning techniques can be significantly affected by carefully-crafted attacks exploiting knowledge of the learning algorithm. They depended on a previously-proposed attack framework to categorize potential attack scenarios against learning-based malware detection tools, by modeling attackers with different skills and capabilities. They then defined and implemented a set of corresponding evasion attacks to thoroughly assess the security of Drebin [12], a machine-learning approach that relied on static analysis for an efficient detection of Android malware directly on the mobile device. Drebin learns a linear SVM classifier. Demontis et al. focused on improving the security of Drebin against stealthier attacks, i.e., carefully-crafted malware samples that avoided detection without exhibiting significant evidence of manipulation. They defined Secure SVM learning algorithm (Sec-SVM) and evaluated the Sec-SVM by testing it under different evasion scenarios. Their experimental results showed that machine learning could be used to improve system security, if one followed an adversary-aware approach that proactively anticipates the attacker. The main contribution of their work is the proposal of a simple and scalable secure-learning paradigm that mitigates the impact of evasion attacks, while only slightly worsening the detection rate in the absence of attack.

Feldman et. al. [21] proposed a system, called Manilyzer, to detect Android malware. The system analyzes the AndroidManifest.xml file and extracts the necessary information from it to determine whether it is malicious or benign. Information used includes permission requests, high priority receivers, and low version numbers. After representing these features in a vector form, detection models are created using Naive Bayes, Support Vector Machine (SVM), K-Nearest Neighbors algorithm (KNN), and C4.5 decision tree algorithm. Accuracy, false positive ratio, and false negative ratio are used for performance comparison. They obtained the result of about 90% accuracy and about 10% false ratio. Feldman et. al. [21] also proposed the addition of network traffic analysis to strengthen their scheme.

Li et. al. [22] proposed a SVM-based malware detection scheme and used both suspicious API calls and risky permission as features. These feature data trains SVM classifier and then the classifier is used to identify unknown malwares in the future. Experiment results showed that the proposed scheme's accuracy is 81% without considering risky permissions and 86% with considering the permissions.

Saracino et. al. [23] proposed a malware detection system for Android devices, called MADAM, which analyzed and correlated features at four levels: kernel, application, user and package. They also classified malware into behavioral classes, each of which performed misbehaviors that characterized them. MADAM detects and effectively blocks more than 96 percent of malicious apps and also shows the low false alarm rate, the negligible performance overhead and limited battery consumption. But MADAM requires to add a monitoring module to a kernel using the `insmod` command. This command and other necessary tools require root access. MADAM, thus, only runs on rooted devices with a kernel having module support.

Xu et. al. [16] proposed ICCDetector that detects malwares based on Inter Components Communication (ICC). They claimed that existing methods utilized many characteristics, but ignored the communications among components within or cross application boundaries. Xu et. al. [16] concentrated on the interaction between components within or cross application boundaries. They also defined five new malware categories according to their ICC characteristics and discovered 43 new malwares by analyzing manually. Their experiment results showed ICCDetector perfroms well.

Dash et. al. [24] showed that it is possible to accurately classify Android malware into families using dynamic analysis only. They also showed that the high level description of the runtime behavior affected the quality of classification. The experiments showed their SVM classifier performed better from 72% to 84%. They also used Conformal Prediction to improve the classification in precision and recall when a sample was poorly differentiated during the training phase.

Arshad et. al. [25] provided the survey on malwares and their penetrations techniques and categorized antimalwares on the basis of detection methods they used. Arshad et. al. [25] presented the concept of hybrid antimalware to address the limitations of existing static and dynamic approaches.

Hou et. al. [15] proposed a new method that used API calls and their relationship with each other. This relationship creates higher level semantics and makes it difficult for attackers to evade the detection. They presented HinDroid, which used a structured heterogeneous information network (HIN) representation of Android apps, and a meta-path-based approach to link the apps. Their experimental results showed that HinDroid outperformed other alternative Android malware detection techniques as well as popular mobile security products.

Li et. al. [26] demonstrated that it was possible to reduce the number of permissions to be analyzed for malware detection, while maintaining high effectiveness and accuracy. Their proposed method was designed to extract only significant permissions through a systematic three-level pruning approach. Their approach can be used effectively by other commonly used supervised learning algorithms.

III. Model Learning

In this section, we present the methodology that we employed in our approach to classify application accesses as malicious or benign. We used a statistical learning method known as Support Vector Machines (SVM) to train the targeted classifier. We first describe the data and then present the classification model learning.

A. Data

The dataset is collected from an Android environment where malicious and benign applications frequently access the system resources through Android API calls. Each of these Android applications referenced in the dataset is previously categorized as either malicious or benign using the antivirus tools provided by VirusTotal. In total, 30,113 malicious and 28,489 benign instances are collected in our dataset.

An instance in the dataset has up to 133,277 attributes, each of which is an API call and is represented by a triple (*API method name, argument, return type*) invoked by an Android application. Hence, the theoretical upper bound of the number of attributes is the number of permutations over APIs, arguments, and return types. Further details on these APIs can be obtained from the Android 7.0 API documentation. Any API triples not accessed by any application are removed from our dataset. The value of an attribute of a data instance is an integer and denotes the number of times the corresponding API triple was invoked by the corresponding application.

In summary, the training dataset is a 58,602 by 133,277 application-api matrix. The dataset is well-balanced between malicious and benign classes. Since an application would invoke a relatively small number of API triples in ordinary scenarios, the matrix is naturally sparse.

Table I shows the API triples that are most commonly accessed by both malicious and benign applications. On the other hand, Table II and Table III present the API triples that are most commonly accessed by malicious apps and benign apps, respectively.

B. Data cleaning and feature reduction

We consider a data instance is noise if it is associated with no API calls. In our dataset, 5.0% or 1,503 instances out of 30,113 malicious instances were noise while there was no noise in the benign instances. In principle, noise should be removed from the training set in order to build a model from truthful information. At the data cleaning stage, we noticed that the access frequency of some APIs were extremely low frequency or even zero. Out of 133,277 attributes, 68.8% or 91,732 attributes show no record of access. These attributes were removed from the training dataset. The removal of such attributes not only improved the quality of the training set and subsequently the performance of the classifier, but also reduced the degree of the curse of high dimensionality. The number of attributes used in our final experiments was 41,545.

C. Classification model

We employed Support Vector Machines (SVM) in our experiment. SVM is commonly defined as a dual optimization problem over the Lagrange multipliers as follows:

$$\max_{\lambda_i} \sum_{i=1}^{n} \lambda_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \lambda_i \lambda_j y_i y_j \langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j)\rangle$$

$$\text{subject to } \sum_{i=1}^{n} \lambda_i y_i = 0,$$

$$0 \le \lambda_i \le C$$

where λ_i is the Lagrange multiplier for the data instance $\mathbf{x}_i$, y_i is the class label of $\mathbf{x}_i$, and $\langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j)\rangle$ is the inner product of the pair of the two data instances, $\mathbf{x}_i$ and $\mathbf{x}_j$. Here, $\phi(\mathbf{x}_i)$ is the transformation of an instance $\mathbf{x}_i$ into a new feature space using the transformation function ϕ. The transformation function facilitates a method for computing similarity between $\mathbf{x}_i$ and $\mathbf{x}_j$ in the respective transformed space more conveniently using the original vectors. This technique is commonly known as a kernel trick. Furthermore, C in the formula above controls a trade-off between maximizing the margin and minimizing the training error [27]. A larger C value tends to reduce the training error while increasing the margin. The dual optimization problem is solved using a quadratic programming problem (QPP) which is integrated into SVM implementations. We used LIBSVM [28] in our experiment.

As the dot product plays a critical role in solving the dual optimization problem, the choice of the transformation function is important. We experimented with the linear kernel, polynomial kernel, radial basis function (RBF) kernel and sigmoid kernel according to their popularity in the literature. They are defined between two data instance vectors $\mathbf{u}$ and $\mathbf{v}$ as follows:

$$\text{Linear kernel } K(\mathbf{u}, \mathbf{v}) = \mathbf{u}^T\mathbf{v}$$

$$\text{Polynomial kernel } K(\mathbf{u}, \mathbf{v}) = (\gamma \mathbf{u}^T\mathbf{v} + c)^d$$

$$\text{Radial Basis Function kernel } K(\mathbf{u}, \mathbf{v}) = \exp\left(-\frac{\|\mathbf{u}-\mathbf{v}\|^2}{2\sigma^2}\right)$$

$$\text{Sigmoid kernel } K(\mathbf{u}, \mathbf{v}) = \tanh(k\mathbf{u}^T\mathbf{v} - \delta)$$

where σ is standard deviation, and γ, c, k, and δ are heuristic coefficients. The RBF kernel has been nominated as the preferred choice in the literature. We compared the performance of these kernel functions in our problem domain. Once a kernel function is chosen, it is apparent that the performance of SVM will depend on the parameters chosen for the kernel function.

IV. Model performance and results

We now present the performance of SVM observed in our experiment in terms of accuracy, recall, precision, specificity, and negative predictive value metrics based on the following four counts:

- True positives (TP): number of malicious instances that are correctly predicted as malicious
- False negatives (FN): number of malicious instances that are incorrectly predicted as benign
- True negatives (TN): number of benign instances that are correctly predicted as benign
- False positives (FP): number of benign instances that are incorrectly predicted as malicious

The definitions of the five performance metrics follow:

Metric	Definition
Accuracy	(TP+TN)/(TP+TN+FP+FN)
Sensitivity or recall	(TP)/(TP+FN)
Specificity	(TN)/(TN+FP)
Precision	(TP)/(TP+FP)
Negative predictive value	(TN)/(TN+FN)

We measured all five of these metrics because each metric has an inherent bias. For example, the accuracy measure is commonly used for classifier performance. However, if the dataset is imbalanced between malicious and benign instances with the overwhelming dominance of the benign instances, the classifier accuracy can be high when benign instances are correctly classified even if the classifier fails to classify malicious ones correctly. For the study of malicious access, recall and precision are as important as accuracy.

TABLE I
THE NAME, ARGUMENTS AND RETURN TYPE OF THE TOP 5 MOST POPULARLY INVOKED APIS

API Method Name	Argument	Return Type
Landroid/os/Parcel;.readValue	Ljava/lang/ClassLoader	Ljava/lang/Object
Ljava/lang/StringBuffer;.wait	JI	V
Ljava/lang/NumberFormatException;.wait	JI	V
Ljava/lang/StringBuilder;.substring	II	Ljava/lang/String
Ljava/lang/StringBuilder;.append	Ljava/lang/String	Ljava/lang/AbstractStringBuilder

TABLE II
TOP 5 MOST FREQUENTLY ACCESSED APIS BY MALICIOUS APPS

API Method Name	Argument	Return Type
Ljava/lang/StringBuilder;.append:	Ljava/lang/String;	Ljava/lang/AbstractStringBuilder;
Ljava/lang/StringBuilder;.substring:	I	Ljava/lang/String;
Ljava/lang/NumberFormatException;.wait:	JI	V
Ljava/lang/StringBuffer;.wait:	JI	V
Landroid/os/Parcel;.readValue:	Ljava/lang/ClassLoader;	Ljava/lang/Object;

TABLE III
TOP 5 MOST FREQUENTLY ACCESSED APIS BY BENIGN APPS

API Method Name	Argument	Return Type
Ljava/lang/StringBuilder;.append:	Ljava/lang/String;	Ljava/lang/AbstractStringBuilder;
Ljava/lang/NumberFormatException;.wait:	JI	V
Landroid/os/Parcel;.readValue:	Ljava/lang/ClassLoader;	Ljava/lang/Object;
Ljava/lang/StringBuilder;.substring:	I	Ljava/lang/String;
Ljava/lang/StringBuffer;.wait:	JI	V

The highest accuracy we obtained is 99.75% using the linear kernel function as shown in Table V. Table V shows the performance metrics and parameter values used in each experiment. The classifier recall of 99.97% using the linear kernel reflects 9 or 0.03% of the 30,113 malicious instances that are incorrectly classified as shown in Table IV.

TABLE IV
CONTINGENCY TABLE FROM LINEAR SVM CLASSIFICATION

		True condition	
		malicious	benign
Prediction	malicious	TP=30,104	FP=138
	benign	FN=9	TN=28,351

In addition, Table V also compares the performance of SVM with different kernel functions. Both RBF (I) and RBF (II) in the table are from the RBF kernel with differently specified parameters. Notice that the poorly chosen parameters for the RBF kernel (RBF (I)) result in a lower accuracy than the accuracy of the polynomial SVM whose parameters were chosen more carefully.

The accuracy 99.75% of the linear SVM is considered to be highly competitive as it was obtained from the balanced dataset between malicious and benign instances. Furthermore, the recall rate of 99.97% of the linear SVM, higher than the accuracy 99.75% of the linear SVM shown in Table V, is encouraging as a failure of detecting a malicious access may not be tolerable in an application domain.

TABLE V
PERFORMANCE COMPARISON OF SVM WITH DIFFERENT KERNELS

Model metric	Sigmoid C=32 $\sigma = 8$	Polynomial C=32 $\sigma = 8$	RBF (I) C=1 σ=144.1	RBF (II) C=32 $\sigma = 8$	Linear C=32 $\sigma = 8$
Sensitivity or recall	91.92%	99.71%	85.50%	99.81%	99.97%
Specificity	91.08%	94.00%	97.54%	98.37%	99.52%
Precision	91.59%	94.60%	97.04%	98.48%	99.54%
Nega. pred. value	91.43%	99.67	87.67%	99.80%	99.97%
Accuracy	91.51%	96.92%	91.68%	99.11%	99.75%

The performance of our approach is compared with other existing work as summarized in Table VI. We adopted this table from [11], and added more recent studies to it. The comparison should be read carefully because different authors used different datasets. Even if the datasets are not identical, they are comparable and hold the information of similar nature. It is encouraging to see that our approach using the linear kernel SVM is steadily in the top tier, and the RBF SVM (II) performs better than other SVM-based approaches.

Additionally, our work obtained the highly competitive results while using the high-dimensional feature set compared to other approaches in Table VI. This fact implies our approach is robust under the curse of dimensionality. Other approaches in Table VI selected a feature set in advance to limit their feature space. However, we experimented with 41,545 features and still achieved very compelling outcomes: 99.75% accuracy and 99.97% recall.

TABLE VI
PERFORMANCE COMPARISON WITH EXISTING WORK
(*THE TABLE IS ADOPTED FROM [11] AND MODIFIED FOR FURTHER INFORMATION)

	Dataset	Features	Classifier	Accuracy
Peiravian & Zhu [19]	1,260 malicious, 1,250 benign	1,326 APIS, 130 Permissions	SVM J48 DT Bagging	93.54-96.88% 92.36-94.46% 93.50-96.39%
Adfer et al. [29]	3,987 malicious 500 benign	169 APIs, 20 API params, Package info.	KNN SVM ID3 DT C4.5 DT	99.1% 96.5% 97.9% 95.5%
Goyal et al. [30]	4,554 malicious, 20,446 benign	300 APIs	Random Forest (trees=10) Linear SVM KNN (k=20)	06.95-99.51% 96.04-97.27% 96.46-98.74%
Alam & Vuong [31]	1,330 malicious, 407 benign	42 dynamic features	Random Forest (trees= 10, 20, 40, 80, 160)	>= 99%
Jung et al. [32]	30,084 malicious, 30,159 benign	50 APIs	Random Forest (trees=10)	97.84-99.98%
Jung et al. [11]	30,084 malicious, 30,159 benign	N popular combined APIs (N=10 to 100)	Random Forest (trees=100) LR k-NN(k=9)	97.46-99.87% 84.11-99.54% 95.24-97.62%
Ours in this paper	30,113 malicious, 28,489 benign	41,545 APIs	Linear SVM	99.75%

V. CONCLUSION

In this work, we focused on the static analysis of Android applications' API calls. We experimented intensively with 58,602 Android applications as well as 133,227 features for malware detection of Android apps using Support Vector Machines (SVM), which showed competitive results compared to existing approaches. Our approach provided 99.75% overall accuracy and a very compelling recall rate of 99.97%. In future work, we will focus on dynamic analysis of Android applications API calls, and the data is inherently considered time series data in such an analysis.

ACKNOWLEDGMENT

We would like to extend our thanks to Hyunjin Kim, Jaemin Jung, Sangchul Han, and Jehee Cho for their helpful discussion about the experimental data as well as their previous work. The research is supported in part by (1) Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Science and ICT (no. 2018R1A2B2004830) and (2) the MSIT (Ministry of Science and ICT), Korea, under the National Program for Excellence in SW supervised by the IITP (Institute for Information communications Technology Promotion)(2017-0-00091).

REFERENCES

[1] "How many phones are in the world?" https://www.bankmycell.com/blog/how-many-phones-are-in-the-world, accessed: 2019-09-10.
[2] "Mobile operating system market share worldwide," https://gs.statcounter.com/os-market-share/mobile/worldwide, accessed: 2019-09-10.
[3] C. Campbell, "Kernel methods: A survey of current techniques," *Neurocomputing*, vol. 48, pp. 63–84, 10 2002.
[4] T. Joachims, "Text categorization with support vector machines: Learning with many relevant features," in *Proceedings of the 10th European Conference on Machine Learning*, ser. ECML'98. Berlin, Heidelberg: Springer-Verlag, 1998, pp. 137–142. [Online]. Available: https://doi.org/10.1007/BFb0026683
[5] S. Tong and D. Koller, "Support vector machine active learning with applications to text classification," *J. Mach. Learn. Res.*, vol. 2, pp. 45–66, Mar. 2002. [Online]. Available: https://doi.org/10.1162/153244302760185243
[6] Y. Liu, K. Wen, Q. Gao, X. Gao, and F. Nie, "Svm based multi-label learning with missing labels for image annotation," *Pattern Recognition*, vol. 78, pp. 307 – 317, 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0031320318300372
[7] M. A. Chandra and S. S. Bedi, "Survey on svm and their application in image classification," *International Journal of Information Technology*, Jan 2018. [Online]. Available: https://doi.org/10.1007/s41870-017-0080-1
[8] W. Yu, T. Liu, R. Valdez, M. Gwinn, and M. Khoury, "Application of support vector machine modeling for prediction of common diseases: The case of diabetes and pre-diabetes," *BMC medical informatics and decision making*, vol. 10, p. 16, 03 2010.
[9] W. K. Resende, R. A. Nascimento, C. R. Xavier, I. F. Lopes, and C. N. Nobre, "The use of support vector machine and genetic algorithms to predict protein function," in *2012 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, Oct 2012, pp. 1773–1778.
[10] J. Jung, K. Lim, B. Kim, S. Cho, S. Han, and K. Suh, "Detecting malicious android apps using the popularity and relations of apis," in *2019 IEEE Second International Conference on Artificial Intelligence and Knowledge Engineering (AIKE)*, June 2019, pp. 309–312.
[11] J. Jung, H. Kim, S. 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, February 2019.
[12] D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, K. Rieck, and C. Siemens, "Drebin: Effective and explainable detection of android malware in your pocket," in *Symposium on Network and Distributed System Security (NDSS)*, vol. 14, 2014, pp. 23–26.
[13] F. Wei, Y. Li, S. Roy, X. Ou, and W. Zhou, "Deep ground truth analysis of current android malware," in *Detection of Intrusions and Malware, and Vulnerability Assessment*, M. Polychronakis and M. Meier, Eds. Cham: Springer International Publishing, 2017, pp. 252–276.

[14] S. Chen, M. Xue, Z. Tang, L. Xu, and H. Zhu, "Stormdroid: A streaminglized machine learning-based system for detecting android malware," in *Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security*, ser. ASIA CCS '16. New York, NY, USA: ACM, 2016, pp. 377–388. [Online]. Available: http://doi.acm.org/10.1145/2897845.2897860

[15] S. Hou, Y. Ye, Y. Song, and M. Abdulhayoglu, "Hindroid: An intelligent android malware detection system based on structured heterogeneous information network," in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2017, pp. 1507–1515.

[16] K. Xu, Y. Li, and R. H. Deng, "Iccdetector: Icc-based malware detection on android," *IEEE Transactions on Information Forensics and Security*, vol. 11, no. 6, pp. 1252–1264, June 2016.

[17] G. Suarez-Tangil, S. K. Dash, M. Ahmadi, J. Kinder, G. Giacinto, and L. Cavallaro, "Droidsieve: Fast and accurate classification of obfuscated android malware," in *Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy*. ACM, 2017, pp. 309–320.

[18] A. Demontis, M. Melis, B. Biggio, D. Maiorca, D. Arp, K. Rieck, I. Corona, G. Giacinto, and F. Roli, "Yes, machine learning can be more secure! a case study on android malware detection," *IEEE Transactions on Dependable and Secure Computing*, vol. 16, no. 4, pp. 711–724, 2019.

[19] N. Peiravian and X. Zhu, "Machine learning for android malware detection using permission and api calls," in *2013 IEEE 25th International Conference on Tools with Artificial Intelligence*, Nov 2013, pp. 300–305.

[20] P. Faruki, V. Ganmoor, V. Laxmi, M. S. Gaur, and A. Bharmal, "Androsimilar: robust statistical feature signature for android malware detection," in *Proceedings of the 6th International Conference on Security of Information and Networks*. ACM, 2013, pp. 152–159.

[21] S. Feldman, D. Stadther, and B. Wang, "Manilyzer: automated android malware detection through manifest analysis," in *2014 IEEE 11th International Conference on Mobile Ad Hoc and Sensor Systems*. IEEE, 2014, pp. 767–772.

[22] W. Li, J. Ge, and G. Dai, "Detecting malware for android platform: An svm-based approach," in *2015 IEEE 2nd International Conference on Cyber Security and Cloud Computing*. IEEE, 2015, pp. 464–469.

[23] A. Saracino, D. Sgandurra, G. Dini, and F. Martinelli, "Madam: Effective and efficient behavior-based android malware detection and prevention," *IEEE Transactions on Dependable and Secure Computing*, vol. 15, no. 1, pp. 83–97, 2016.

[24] S. K. Dash, G. Suarez-Tangil, S. Khan, K. Tam, M. Ahmadi, J. Kinder, and L. Cavallaro, "Droidscribe: Classifying android malware based on runtime behavior," in *2016 IEEE Security and Privacy Workshops (SPW)*. IEEE, 2016, pp. 252–261.

[25] S. Arshad, M. A. Shah, A. Khan, and M. Ahmed, "Android malware detection & protection: a survey," *International Journal of Advanced Computer Science and Applications*, vol. 7, no. 2, pp. 463–475, 2016.

[26] J. Li, L. Sun, Q. Yan, Z. Li, W. Srisa-an, and H. Ye, "Significant permission identification for machine-learning-based android malware detection," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 7, pp. 3216–3225, 2018.

[27] P.-N. Tan, M. Steinbach, A. Karpatne, and V. Kumar, *Introduction to Data Mining, second edition*, 2nd ed. New York, NY: Pearson, 2019.

[28] C.-C. Chang and C.-J. Lin, "Libsvm: A library for support vector machines," *ACM Trans. Intell. Syst. Technol.*, vol. 2, no. 3, pp. 27:1–27:27, May 2011.

[29] Y. Aafer, W. Du, and H. Yin, "Droidapiminer: Mining api-level features for robust malware detection in android," vol. 127, 09 2013, pp. 86–103.

[30] R. Goyal, A. Spognardi, N. Dragoni, and M. Argyriou, "Safedroid: A distributed malware detection service for android," in *2016 IEEE 9th International Conference on Service-Oriented Computing and Applications (SOCA)*, Nov 2016, pp. 59–66.

[31] M. S. Alam and S. T. Vuong, "Random forest classification for detecting android malware," in *Proceedings of the 2013 IEEE International Conference on Green Computing and Communications and IEEE Internet of Things and IEEE Cyber, Physical and Social Computing*, ser. GREENCOM-ITHINGS-CPSCOM '13. Washington, DC, USA: IEEE Computer Society, 2013, pp. 663–669. [Online]. Available: http://dx.doi.org/10.1109/GreenCom-iThings-CPSCom.2013.122

[32] J. Jung, H. Kim, D. Shin, M. Lee, H. Lee, S. Cho, and K. Suh, "Android malware detection based on useful api calls and machine learning," in *2018 IEEE First International Conference on Artificial Intelligence and Knowledge Engineering (AIKE)*, Sep. 2018, pp. 175–178.