Intelligent Malware Detection Based on Hybrid Learning of API and ACG on Android

Kichang Kim, Eunbyeol Ko, Jinsung Kim, and Jeong Hyun Yi*
School of Software, Soongsil University, Seoul, 06978, Republic of Korea
{kckim7008, kongstar159, okokabv}@soongsil.ac.kr, jhyi@ssu.ac.kr

Abstract

Mobile devices will continue to be central in providing personalized services in the hyper-connected era following the introduction of 5G network services. If a mobile device is exposed to malwares, there is a risk of malware spreading to all the devices it is connected to in an instant. For example, malware can transit from mobile devices to autonomous vehicles that share data through various sensors and that are hyper-connection capable with a server or other device on a 5G network. It is thus becoming more important to preemptively anticipate the behavior of mobile malware using machine learning techniques based on pre-learned datasets. In this paper, we propose a scheme to identify malicious codes by extracting `APIs` used in Android apps by hybridizing machine learning techniques based not only on `APIs` but also `ACG`. The proposed scheme aims to reduce false positives of existing approaches using only `APIs` and improving performance problems of `ACG` approaches using excessive features. In addition, we evaluate the performance of the proposed scheme by comparing and analyzing the experimental results of the proposed scheme and the existing schemes for third-party malicious code samples.

Keywords: Malware Detection, Machine Learning, API, API Call Graph

1 Introduction

Along with Android's high market share, new applications are developed every day, and malwares targeting vulnerabilities in Android applications is also on the rise. However, it is not easy for general users to cope with the Android characteristics, in which it is easy to acquire and redistribute code. Thus, it is essential to develop technology that identifies malicious behavior by accurately analyzing application operations.

In addition, with the commercialization of 5G networks that can hyper-connect all components in the IoT era, billions of mobile devices will share data to provide various user-centric services [10]. These hyper-connected and personalized services will still be provided by mobile devices, which if exposed to malwares, run the risk of being infected by malware. For example, consider that malware transits from mobile devices to autonomous vehicles that share data through various sensors that are hyper-connection capable with a server or other devices on a 5G network. Therefore, in the future, it is more important to preemptively predict the response using the machine learning algorithms based on the pre-learned dataset rather than detecting malicious activity after the mobile device is infected.

In order to detect such mobile malicious codes, techniques that mainly use permissions [20] and descriptions [15] have been proposed. However, these techniques rely on subjective judgments or user feedback and have limitations in analyzing malicious behaviors. Recently, a variety of malware detection techniques have been studied. In particular, techniques for determining malicious behavior based on

Journal of Internet Services and Information Security, volume: 9, number: 4 (November 2019), pp. 39-48
*Corresponding author: Jeong Hyun Yi, School of Software, Soongsil University, Seoul, Korea, 06978, Tel: +82-2-820-0914, email: jhyi@ssu.ac.kr

Application Programming Interface (API) information used in an application along with machine learning technology have been actively researched [6, 24, 21].

Therefore, in this paper, we propose a scheme to identify malicious codes by hybridizing machine learning techniques based on API and API Call Graph (ACG) to Android, which has the largest market share in mobile devices. The proposed scheme is expected to provide a foundation for faster and more accurate malware prediction by reducing the **false positive rate** of a single API and improving the **performance degradation** of the ACG technique due to the excessive use of features.

This paper is organized as follows. Section 2 describes the related work, and Section 3 describes the proposed scheme. In Section 4, we report on the implementation of the proposed scheme, compare the performance with the existing schemes, and provide our conclusions in Section 5.

2 Related Work

API based malware detection technology can be classified into two phases: learning and decision. Among these, the techniques can be classified into API based approach and ACG based approach according to API extraction and modeling methods. Finally, classification techniques applied in common to both approaches. This section outlines the two approaches.

2.1 API based Approach

This approach is used to predict whether an application is malicious by extracting API from malware and modeling it. Examples of such approach are given in Yerima, et al. [23], Qiao, et al. [17], and DroidAPIMiner [3]. For example, analyzing the scheme proposed by Yerima, et al., in this scheme, first, APIs and all permissions used in the target application are extracted, and the frequency of occurrence is measured. The *Mutual Information (MI)* score [8] is then measured based on this. The MI score is an indicator of the interdependence between two variables using probability. Given $C = \{Benign, Malicious\}$, the following equation quantifies the interdependency of each API.

$$\mathtt{MI}(C, Xi) = \sum_{c \in \{mal, ben\}} \sum_{x \in \{0,1\}} P(C = c, Xi = x) \cdot \log_2 \left(\frac{P(C = c, Xi = x)}{P(C = c) \cdot P(Xi = x)} \right)$$

In this equation, $P(C = c)$ is the probability that APK is *Benign* or *Malicious*. $P(X_i = x)$ is the probability of appearance of the API, where $x = 0$ is the probability that API does not exist, and $x = 1$ is the probability that API exists. And $P(C = c, X_i = x)$ is the probability that both scenarios occur simultaneously, i.e., the probability of API exists and being malicious. In this way, a total of four probability calculations can be performed, and the sum of all of them is the value of MI. The larger the value of the MI, the more information about the group is contained in the entity.

Table 1 shows the top 10 MI scores among the information measured by Yerima, et al. This scheme uses the permission and API with a high MI score as features using a Bayesian-based algorithm. It also specifies three models and then calculates the accuracy.: a model using only permission, a model using only code-specific information including API, and a combination of both models.

Although the scheme proposed by Yerima, et al. is very simple and efficient, it detects with a single API and is thus limited when detecting malicious behavior. The same API may or may not be considered malicious, depending on how it is used. For example, if a sensitive API that accesses personal information is called, it will be judged to be malicious if it is connected to an API that transmits the information to another place over the network. However, if it operates only within the device, it cannot be judged to be malicious. Thus, rather than simply relying on the use of a particular API, more precise detection is possible through analysis of how the API was used.

Table 1: MI Scoring Example of [23]

Permissions and Properties	Benign	Malicious	Total	MI Score
getSubscriberId(TelephonyManager)	42	742	784	0.42853
READ_SMS	20	591	611	0.32920
WRITE_SMS	11	466	477	0.25053
getDeviceId(TelephonyManager)	316	854	1170	0.22919
READ_PHONE_STATE	388	888	1276	0.20962
SEND_SMS	24	443	467	0.20709
getSimSerialNumber(TelephonyManager)	35	455	490	0.19674
RECEIVE_SMS	14	394	408	0.19305
.apk	89	537	626	0.18202
chmod	19	389	408	0.17989
...				

2.2 ACG based Approach

MaMaDroid [13], Peiravian and Xingquan [16], and Droidmat [22] are techniques used to determine whether malware is involved in the call relationship between APIs, rather than targeting APIs extracted from malware. As a representative example, MaMaDroid is described as follows. First, define API by dividing it into family and package as shown in Figure 1, and extract ACG using Flowdroid [4] to create a sequence of family and package. After that, the training data set is created through modeling using the Markov chain, and the usage pattern of API call sequence is used for learning. For classification, $k-NN$ $(k-NearestNeighbors)$ [11] and Random Forest [5] classifiers are used.

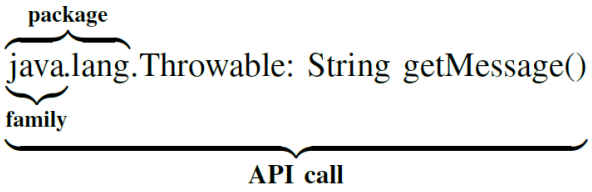

Figure 1: Definition of API Call

MaMaDroid counts and models all the ACGs used in the app, with the ACG being used as a feature ranging from hundreds of thousands to millions, and counting the number of occurrences, which involves a great deal for data preprocessing and classification, and is a lengthy process. Too many features can reduce not only classification speed but also so classification accuracy, so proper feature sizing can improve classification performance.

3 Proposed Scheme

This section addresses the problems of using a single API in the API based approach and the problems of using too many features in the ACG based approach. A hybrid approach that takes advantage of the two approaches is also proposed. Figure 3 shows the overall structure of the proposed scheme. The proposed scheme is largely divided into the learning phase and decision phase.

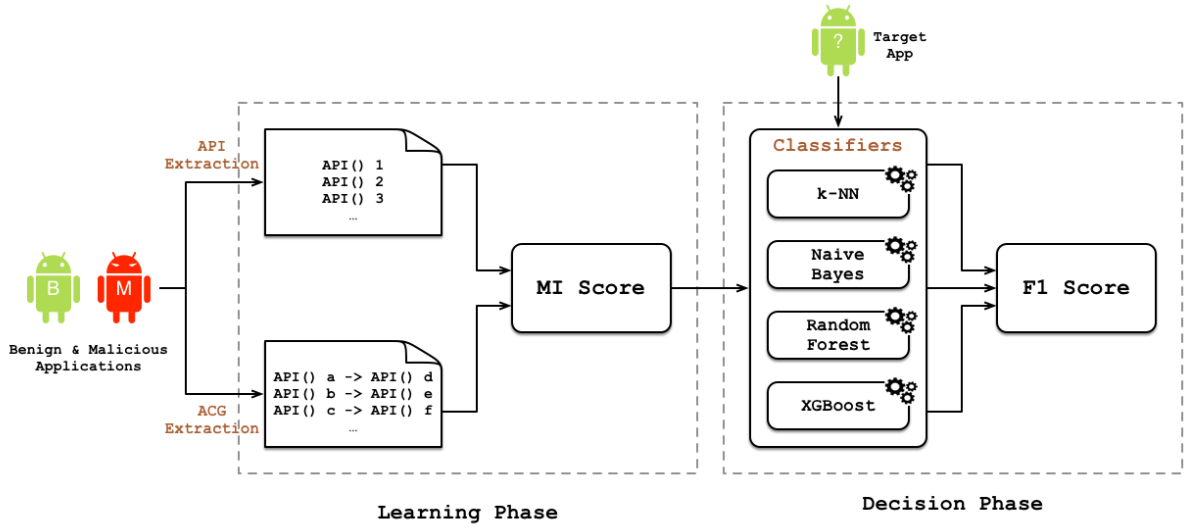

Figure 2: System Architecture of Proposed Scheme

3.1 Learning Phase

This phase extracts the APIs and ACGs used by the benign applications and the malware set, respectively, to analyze the usage pattern of the API in the application.

First, the API extraction process is as follows. The API can be extracted by parsing the classes.dex file of the application. The DEX file consists of 8 fields, among which method related data is included in the method table and class def table. The class def table is a list of user-defined classes. Among them, class data item includes direct method and virtual method information. Direct methods contain information indicating the offset of the method, which points to the index of the method table. After reaching the string table through the name index value, which is obtained by calculating the difference between the method index offsets of the direct and virtual methods in the method table, it is possible to finally find and extract the API name from the data section. In this way, the package, class, and method information of the API can be obtained as shown in Table 2.

Table 2: Extracted API Example

API
android/transition/TransitionManager;beginDelayedTransition
android/os/Parcel;writeStringList
android/view/View;getWidth
java/lang/reflect/Field;get
android/net/NetworkInfo;getExtraInfo
android/view/View;getId
...

Next, ACG can be extracted using Taint analysis [14]. This paper also uses Flowdroid just as it did for MaMaDroid. Flowdroid is a framework that provides Taint analysis based on Soot [19], a Java optimization framework. It extracts information by analyzing AndroidManifest.xml, classes.dex, layout.xml files of Android APK, and extracts ACG by analyzing the flow from source to sink in the main method. The API sequence generated through the extracted ACG has the form shown in Fig-

ure 3. In this paper, the package unit of API is used for analysis, so it is finally used in the form of "bighead.wallpaper.beauty To java.lang".

```
specialinvoke $r0.<java.lang.Object: void <init>()>() in
<bighead.wallpaper.beauty.Wallpaper: void <clinit>()> ==> <java.lang.Object: void <init>()>
```

Figure 3: Extracted ACG Example

Once the APIs and ACGs extraction is complete, the frequency used by the malware set and the benign application set is measured. The MI score is calculated using the frequency. As explained in Section 2.2, MaMaDroid takes a lot of time to learn and classify using all of the hundreds of thousands of ACGs extracted, while the proposed scheme calculates the MI score of the extracted ACGs for more efficient learning and classification, and selects only the ACG pattern that is used mainly in malwares as the feature. Because we are aiming at detecting malwares, we generate MI scores against APIs that are used more often in malwares than in benign applications.

For example, the MI score of the android/telephony/TelephonyManager;getSubscriberId API is determined as follows. It is assumed that the number of benign and malicious apps used is 1000 and 2000, respectively, and the APIs are found in 106 benign apps and 1207 malicious apps.

$$\begin{aligned}
\mathtt{MI}(C,Xi) &= P(mal,Xi=1)\cdot\log_2\left(\frac{P(mal,Xi=1)}{P(mal)\cdot P(Xi=1)}\right)+P(ben,Xi=1)\cdot\log_2\left(\frac{P(ben,Xi=1)}{P(ben)\cdot P(Xi=1)}\right)\\
&+P(mal,Xi=0)\cdot\log_2\left(\frac{P(mal,Xi=0)}{P(mal)\cdot P(Xi=0)}\right)+P(ben,Xi=0)\cdot\log_2\left(\frac{P(ben,Xi=0)}{P(ben)\cdot P(Xi=0)}\right)\\
&=\frac{1207}{3000}\cdot\log_2\frac{\frac{1207}{3000}}{\frac{2000}{3000}\cdot\frac{1313}{3000}}+\frac{106}{3000}\cdot\log_2\frac{\frac{106}{3000}}{\frac{1000}{3000}\cdot\frac{1313}{3000}}\\
&+\frac{793}{3000}\cdot\log_2\frac{\frac{793}{3000}}{\frac{2000}{3000}\cdot\frac{1687}{3000}}+\frac{894}{3000}\cdot\log_2\frac{\frac{894}{3000}}{\frac{1000}{3000}\cdot\frac{1687}{3000}}=0.180272
\end{aligned}$$

3.2 Decision Phase

The F1 score [18] of the target app was calculated using the typical machine learning algorithms $k-NN$, Naïve Bayes [9], Random Forest, and XGBoost [7]. The performance of the classifier is verified through 10-fold cross validation [12]. F1-Score is defined as follows.

$$\mathtt{F1\text{-}Score}=2\cdot\frac{\mathtt{precision}\cdot\mathtt{recall}}{\mathtt{precision}+\mathtt{recall}}$$

where precision is $TP/(TP+FP)$ and recall is $TP/(TP+FN)$, and also TP, FP, and FN indicate true positive, false positive, and false negative, respectively.

4 Experiments

In order to evaluate the performance of the proposed scheme, we implement the Yerima, et al., MaMaDroid on the same training dataset and analyze the accuracy of the proposed scheme, comparing with them.

4.1 Experimental Setup

The proposed scheme was implemented using Python 3.7.2, and the experiment was conducted on a Windows 10 with Intel i5-7400 CPU, and 32GB RAM. Malware samples used in the experiment were 2000 obtained from Virusshare [2], and 1000 benign applications used in the Google Play Store [1].

4.2 Experiment Results

Experimental results of each phase of the proposed scheme are described separately.

4.2.1 Learning Phase

First, to create an API-based model, classes.dex file is parsed and extracted to the method unit of API, and MI score ranking of all APIs is generated, and 40 APIs with the highest scores are used as the features. Generate training and test set by vectorizing the presence or absence of 40 APIs to 1 or 0. Table 3 shows the API MI scores of 40 APIs.

Table 3: API based MI Scoring Results

No.	API	Malicious	Benign	MI Score
1	Landroid/telephony/TelephonyManager;getSubscriberId	1207	106	0.180272
2	Landroid/net/NetworkInfo;getExtraInfo	1118	81	0.175569
3	Landroid/app/Notification;setLatestEventInfo	1393	251	0.132037
4	Landroid/telephony/SmsManager;sendTextMessage	708	27	0.118491
5	Landroid/telephony/gsm/GsmCellLocation;getLac	797	49	0.117406
6	Landroid/app/Activity;onKeyDown	1446	304	0.11694
7	Landroid/telephony/SmsManager;getDefault	719	33	0.114415
8	Landroid/telephony/gsm/GsmCellLocation;getCid	798	54	0.112885
9	Landroid/telephony/TelephonyManager;getCellLocation	800	55	0.112451
...				
40	Landroid/util/FloatMath;sqrt	406	41	0.039788

Next, extract API call sequence using Flowdroid to create ACG. In this paper, we use the information up to the API package unit defined by Google and calculate the MI score to use the top 40 API call sequences as a feature. Table 4 shows 40 ACGs and their respective MI scores.

Table 4: ACG based MI Scoring Results

No.	ACG	Malicious	Benign	MI Score
1	android.util To java.lang	1827	900	0.033248028
2	Selfdefined To android.telephony.cdma	311	30	0.023368218
3	java.util To android.webkit	168	4	0.022288056
4	android.telephony.cdma To java.lang	301	30	0.021959577
5	android.net To android.net	1828	877	0.018417745
6	java.io To java.lang	1935	901	0.017549512
7	org.apache.http.conn.scheme To java.lang	851	219	0.017139855
8	Selfdefined To android.util	1830	871	0.01543745
9	Obfuscated To obfuscated	1935	897	0.015065262
...				
40	android.widget To android.os	31	0	0.005214148

Finally, in the proposed hybrid model, the top 20 features were extracted from the two results presented above. A new dataset was constructed using 40 features. Table 5 shows the 40 features used in the proposed scheme.

Table 5: MI Scoring Results for Proposed Scheme

No.	API + ACG	Malicious	Benign	MI Score
1	Landroid/telephony/TelephonyManager;getSubscriberId	1207	106	0.180272
2	Landroid/net/NetworkInfo;getExtraInfo	1118	81	0.175569
3	Landroid/app/Notification;setLatestEventInfo	1393	251	0.132037
4	Landroid/telephony/SmsManager;sendTextMessage	708	27	0.118491
	...			
20	Landroid/widget/RemoteViews;getLayoutId	379	4	0.071514
21	android.util To java.lang	1827	900	0.033248028
22	Selfdefined To android.telephony.cdma	311	30	0.023368218
23	java.util To android.webkit	168	4	0.022288056
24	android.telephony.cdma To java.lang	301	30	0.021959577
	...			
40	android.app To android.widget	84	5	0.00800451

4.2.2 Decision Phase

After the MI scoring was obtained, the accuracy of the proposed scheme was compared with Yerima et al., and MaMaDroid, using 3-NN, Naïve Bayes (NB), XGBoost (XGB), and Random Forest (RF) classifiers. F1-Score measurement results are shown in Table 6, and Figure 4 is a graphical representation.

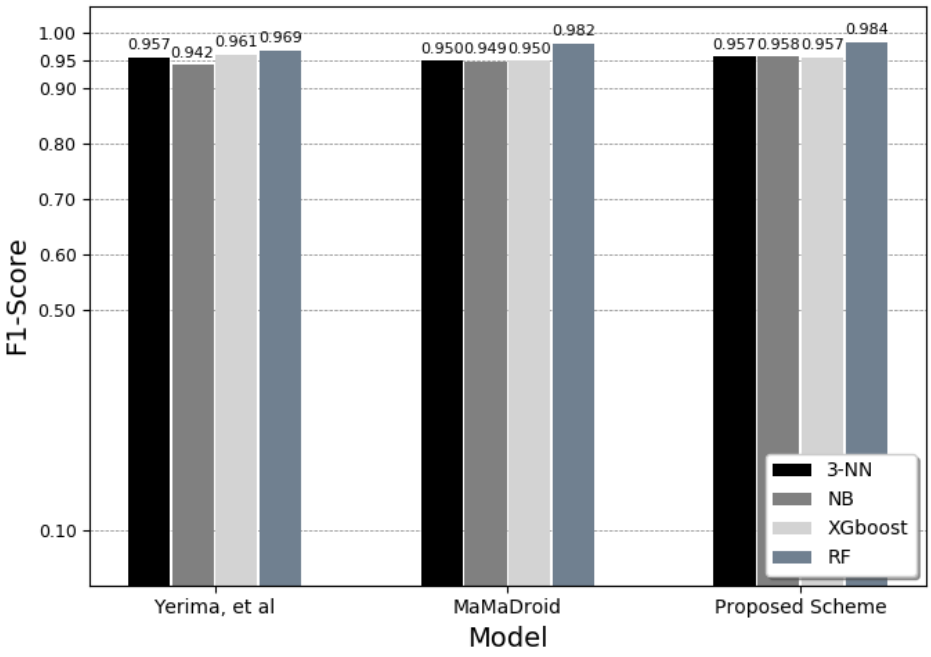

Figure 4: Classification Performance Comparison

Table 6: F1-Score of Three Models

Classification / Model	F1-Score			
	3-NN	NB	XGB	RF
Yerima, et al	0.957	0.942	0.961	0.969
MaMaDroid	0.950	0.949	0.950	0.982
Proposed Scheme	0.957	0.958	0.957	0.984

Experimental results show that the accuracy of the proposed scheme is high among the three schemes, and the accuracy is highest when the Random Forest classifier is used. In addition, the proposed scheme took about 8 seconds to classify, while the MaMaDroid took about 24,350 seconds. Accordingly, it can be seen that the hybrid model proposed in this paper was more efficient at classifying malwares.

5 Conclusion

In this paper, we propose a scheme to reduce the error of existing API based schemes and to improve the performance degradation of ACG based schemes by exploiting the advantages of API and ACG based approaches. Our results confirmed that hybrid learning can detect malicious behavior more accurately than when API or ACG are used alone.

The use of more sample applications and improved machine learning algorithms will make detection more accurate, and we expect to make meaningful contributions to the field of application security research in the future.

Acknowledgements

This research was supported by the Mid-Career Researcher program through the National Research Foundation of Korea (NRF) funded by the MSIT (Ministry of Science and ICT) (NRF-2017R1A2B4008822).

References

[1] "Google Play Store," https://play.google.com/store/apps, accessed on: Jul. 2019.

[2] "VirusShare," https://virusshare.com/, accessed on: Jul. 2019.

[3] Y. Aafer, W. Du, and H. Yin, "DroidAPIMiner: Mining APILevel Features for Robust Malware Detection in Android," in *International Conference on Security and Privacy in Communication Networks*, 2013, pp. 86–103.

[4] S. Arzt, S. Rasthofer, C. Fritz, E. Bodden, A. Bartel, J. Klein, Y. L. Traon, D. Octeau, and P. McDaniel, "FlowDroid: Precise Context, Flow, Field, Object-sensitive and Lifecycle-aware Taint Analysis for Android Apps," in *Proc. of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI'14), Edinburgh, United Kingdom*. ACM New York, June 2014, pp. 259–269.

[5] L. Breiman, *Machine Learning*. Springer, 2001, vol. 45.

[6] S. Chen, M. Xue, Z. Tang, L. Xu, and H. Zhu, "StormDroid: A Streaminglized Machine Learning-Based System for Detecting Android Malware," in *Proc. of the 11th ACM on Asia Conference on Computer and Communications Security (ASIA CCS'16), Xi'an, China*. ACM New York, May 2016, pp. 377–388.

[7] T. Chen and C. Guestrin, "XGBoost: A Scalable Tree Boosting System," in *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2016, pp. 785–794.

[8] T. M. Cover and J. A. Thomas, *Elements of Information Theory*. Wiley, 1991.

[9] E. Frank and R. R. Bouckaert, "Naive Bayes for Text Classification with Unbalanced Classes," in *European Conference on Principles of Data Mining and Knowledge Discovery in Databases*, 2006, pp. 503–510.

[10] G. Group, "Forecast: A Global Internet of Things 2014-2020," November 2015.

[11] J. D. Kelly and L. Davis, "Hybridizing the Genetic Algorithm and the k Nearest Neighbors Classification Algorithm," in *International Conference on Genetic Algorithms*, 1991, pp. 377–383.

[12] R. Kohavi, "A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection," in *International Joint Conference on Artificial Intelligence*, 1995, pp. 1137–1143.

[13] E. Mariconti, L. Onwuzurike, P. Andriotis, E. D. Cristofaro, G. Ross, and G. Stringhini, "MaMaDroid: Detecting Android Malware by Building Markov Chains of Behavioral Models," in *Network and Distributed System Security Symposium*, 2017.

[14] J. Newsome and D. X. Song, "Dynamic Taint Analysis for Automatic Detection, Analysis, and Signature Generation of Exploits on Commodity Software," in *Network and Distributed System Security Symposium*, 2005.

[15] R. Pandita, X. Xiao, W. Yang, W. Enck, and T. Xie, "WHYPER: Towards Automating Risk Assessment of Mobile Applications," in *The 22nd USENIX Security Symposium*, 2013, pp. 527—542.

[16] N. Peiravian and X. Zhu, "Machine Learning for Android Malware Detection Using Permission and API Calls," in *Proc. of the IEEE 25th International Conference on Tools with Artificial Intelligence, Herndon, VA*. IEEE, November 2013, pp. 300–305.

[17] M. Qiao, A. H. Sung, and Q. Liu, "Merging Permission and API Features for Android Malware Detection," in *International Conference on Advanced Applied Informatics*, vol. 5, 2016, pp. 566–571.

[18] C. J. V. Rijsbergen, *Information Retrieval (2nd ed.)*. Butterworth-Heinemann, 1979.

[19] R. Vallee-Rai, P. Co, E. Gagnon, L. Hendren, P. Lam, and V. Sundaresan, "Soot - A Java Bytecode Optimization Framework," in *Conference of the Centre for Advanced Studies on Collaborative Research*, 1999, p. 13.

[20] Y. Wang, J. Zheng, C. Sun, and S. Mukkamaka, "Quantitative Security Risk Assessment of Android Permissions and Applications," in *IFIP Annual Conference on Data and Applications Security and Privacy XXVII*, vol. 7964, 2013, pp. 226–241.

[21] Westyarian, Y. Rosmansyah, and B. Dabarsyan, "Malware Detection on Android Smartphones Using API Class and Machine Learning," in *International Conference on Electrical Engineering and Informatics*, 2015, pp. 294–297.

[22] D.-J. Wu, C.-H. Mao, T.-E. Wei, H.-M. Lee, and K.-P. Wu, "DroidMat: Android Malware Detection Through Manifest and API Calls Tracing," in *Asia Joint Conference on Information Security*, 2012, pp. 62–69.

[23] S. Y. Yerima, S. Sezer, and G. McWilliams, "Analysis of Bayesian Classification-based Approaches for Android Malware Detection," *IET Information Security*, vol. 8, pp. 25–36, 2014.

[24] X. Zhang and Z. Jin, "A New Semantics Based Android Malware Detection," in *Proc. of the 2nd IEEE International Conference on Computer and Communications (ICCC'16), Chengdu, China*. IEEE, October 2016, pp. 1412–1416.

Author Biography

Kichang Kim received his B.S. degree in Mathematics and 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.

Eunbyeol Ko received her B.S. degree in Mathematics from Soongsil University, Seoul, South Korea in 2019. She is a M.S. student in Software, Soongsil University. respectively. Her research interests include mobile application security, mobile platform security, and machine learning.

Jinsung Kim received his B.S. degree in degree in Computer Science and Engineering from Korea National University of Transportation, Seoul, South Korea in 2018. He is a M.S. student in Software Convergence, Soongsil University. respectively. His 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, Seoul. His research interests include mobile security and privacy, IoT security, and applied cryptography.