SVM-based Android Application Classification using API Calls

Myeonggeon Lee[1], Masoud Reyhani Hamedani[1], Gyoosik Kim[1], Tae Kyung Kim[2], and Seong-je Cho[1*]
[1]Dept. of Computer Science and Engineering, Dankook University
Yongin-si, Gyeonggi-do 16890 South Korea
{mglee, masoud, erewe4, sjcho}@dankook.ac.kr
[2]EduAI center, Dankook University
Dankook University, Yongin-si, Gyeonggi-do, 16890, South Korea
kimtk@dankook.ac.kr

Abstract

Measuring similarity between a suspicious app and the existing ones in the app store is one of the existing mechanisms to make the app stores healthy and sustainable against pirated apps, repackaged apps, and malware. To improve the efficiency of similarity computation between apps, it is essential to reduce the number of apps to be compared with the suspicious one; the app classification is one of the methods to achieve this by detecting a possible category for the suspicious app and measuring the similarity between the suspicious app and only the apps in that category. In this paper, we propose a technique for app classification by applying support vector machine (SVM). We extract API calls as a feature for classification from the apps bytecode, consider two candidate features, classes and methods related to API calls, and carefully analyze which of them is more beneficial for app classification. In order to evaluate our technique, we conduct extensive experiments with a real-world dataset.

Keywords: Application classification, API call, SVM, Android app, App similarity

1 Introduction

Among existing smartphone operating systems such as Android, BlackBerry, iOS, and Windows Mobile, Android is the most widely used in the market. There are millions of Android applications (in short, apps) in official Google Play Store[3] and third-party stores; on February 2017, there were 2.7 million apps on Google Play Store, and more than 600,000 apps on Amazon App Store[1, 10]. The store managers cannot adequately review and check all the released apps in the store; more specifically, there are a large number of pirated or malicious apps available in the stores[11]. The apps are easy to be reverse-engineered and repackaged due to Java bytecode characteristics. By taking advantage of this property, attackers reverse engineer benign apps, insert pirated or malicious codes into the original codes, repackage the apps, and redistribute them through the app stores or blog.

Therefore, detecting pirated software, variants of existing malware, and repackaged apps are some of the serious problems in app stores where computing the similarity between apps is one of the key techniques to solve the aforementioned problems.

A similarity computation method measures the similarity by comparing the features extracted from a benign app with the features extracted from a suspicious app to decide whether the suspicious app is an

Research Briefs on Information & Communication Technology Evolution (ReBICTE), Vol. 3, Article No. 18 (November 15, 2017)
*Corresponding author: Department of Computer Engineering, Dankook University, Yongin-si, Gyeonggi-do, 16890, Korea, Tel: +82-31-8005-3239

illegal copy of the benign app or a malicious one. For similarity computation, it is important to reduce the number of apps to be compared with the suspicious app in order to improve the efficiency and scalability. One effective strategy is to classify apps in bytecode or binary level[4, 12, 13, 14].

In this paper, we propose an effective method for app classification based on the support vector machine (SVM) where the API calls are used as a feature. Our motivation is that API calls is helpful and unique information, which clearly represent the app's functionalities and behaviors [7, 15]; for example, an app cannot read SMS or access to the camera without invoking particular APIs. We demonstrate the effectiveness of our method by conducting extensive experiments with a real-world dataset, which is constructed based on Google Play Store.

2 Related Work

In reference[13], the API calls, strings, and URL information are extracted from the apps as the features for classification. For each feature, the apps are ranked according to which categories they belong to, and the categories were grouped together. Although this study considers the possibility of classifying the apps based on the static features extracted from the apps themselves, the size of the experimental dataset is too small (i.e., only 12 apps) and experiments are only conducted with the "weather" category. In reference[14], 42 apps are categorized into 11 categories where strings are utilized as the feature. In the preprocessing step, extracted strings are indexed by using information retrieval (IR) techniques, and important words are selected for each category through weighting, threshold, and Latent Dirichlet Allocation (LDA) analyzing. Using the selected words, the apps are classified into different categories by measuring the similarity of the apps. The classification result are compared with the manual analysis of the apps. In reference [12], the apps are classified by applying machine learning techniques to both strings and permissions as the features. The employed dataset contains 820 apps, which are categorized into seven categories. The classification is performed and evaluated by utilizing four different machine learning techniques as Naive Bayesian, Decision Trees, K-Nearest Neighbor (KNN), and Support Vector Machine (SVM). Reference [4] proposes Naive Bayesian modeling based on the permissions as the feature to classify apps. Through this, a comparison is made between malicious and benign apps.

Androguard is an open-source and static analysis tool for reverse engineering Android apps which can create control flow graphs for each method [5, 6]. It can find similarities and differences in code between two apps by computing Normalized Compression Distance (NCD) between each method pairs. It then shows IDENTICAL, SIMILAR, NEW, DELETED and SKIPPED methods of first app compared to another one. Androidguard can also find similarities and differences between two methods by converting each unique instruction in basic block of methods into a string and then applying Longest Common Subsequence (LCS) algorithm on these strings of two basic blocks. In addition, Androgurad can used to detect a repackaged app.

Juxtapp is a static and scalable infrastructure to analyze code similarity of Android apps [8]. It can group the unlabeled classes.dex files of apps based on some predefined criteria, in order to reduce the comparison overhead. As feature for the criteria, Juxtapp considered k-grams of various opcode sequence patterns within each basic block of the app. It can determine whether apps contain copies of buggy code, and detect various code reuse cases such as software piracy, instances of known malware, and code repackaging. This work has a limitation that it does not perform evaluation in terms of false negative rate [9].

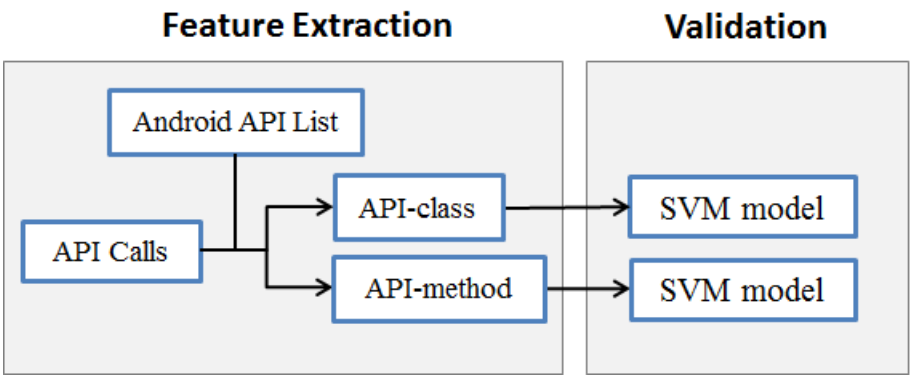

Figure 1: An overview of our proposed method.

3 Proposed Method

The Android platform provides a collection of APIs that are used by apps to interact with the underlying Android system and the smartphone[7]. Android developers write apps by invoking appropriate APIs to implement the functionalities of the apps; API calls can clearly capture the app's behaviors and functionalities [7]. Therefore, in our method, we utilize the API calls as a useful feature for classifying apps based on their functionalities. As shown in Figure 1, we consider API-classes and API-methods as two candidate features associated with the API calls. In this paper, the API-classes and API-methods mean the Java classes and methods used in the target Android apps respectively. Then, SVM is applied to both of the two candidate features. Through experiments, the one feature of them showing the better performance in app classification is being finally selected as the representative feature of the API call information.

3.1 Feature Extraction

We extract the API calls based on the information obtained from app itself and the Android platform. A typical app contains different files and folders that are packaged into a single Android Package (APK), an archive file type easily extractable by any archiving software [7]. The classes.dex is an important file in an APK that contains the app's bytecode where the API calls can be minded from; the classes.dex file contains identifiers for all the methods, user-defined methods and APIs, referred to by the app. We also construct a list of all the available APIs in the Android platform by crawling the Android Developers Site[2]. By using the constructed Android API list, we filter out the list of all the available methods in the app as follows. We neglect the user-defined methods and support libraries for the following reasons. The user-defined methods do not contribute to the app classification because they have their own names selected by the developer, and the support libraries are widely used by lots of apps regardless of the apps functionalities. The API-class candidate contains only the names of APIs' owner classes, while API-method candidate contains the APIs themselves.

Table 1 shows an abstract view of our dataset. Each row represents an app and contains its name, category, and feature values as api_1 to api_n; if the app contains feature value api_i, the content of the corresponding column will be 1, otherwise 0.

3.2 Analytical Method

We apply SVM to both API-class and API-method candidates to analyze their effectiveness in apps classification. We note that in our dataset, the apps are assigned to categories based on their functionalities. Therefore, it is possible that more than one category are assigned to some apps; for example, an app that belongs to the 'News & Magazines' category can also belong to the 'Books & Reference' category. Therefore, for each app, we select the top-*k* appropriate categories.

4 Experimental Evaluation

Table 2 represents the statistics about our dataset collected from Google Play between December 2016 and July 2017; there are 4,724 apps, which are divided into 12 categories. For the case of the dataset, the number of classes and methods used as features is 2,286 and 30,588 respectively. In order to provide a comparison of two features, we have classified Android apps using SVM classifier implemented in *scikit-learn*, a popular machine learning library. *Scikit-learn* is an open source machine learning library written in the Python programming language. App classification has been performed by the SVM with a Radial Basis Function (RBF) kernel.

Table 3 shows the results of classification with the API-class feature. In the verification process, the accuracy is calculated by applying the 10-fold cross validation on top-*k* categories. When *k* is 1, the accuracy is 41.95%; however, as *k* increases, the accuracy increases sharply as well. The reason is that there are some apps belonging to more than one category as explained in Section 3.2.

Table 4 shows the results of classification with the API-method feature. As *k* increases, the accuracy also increases as observed with the API-class feature.

Figure 2 illustrates the accuracy of classification with both features on *k* as 1, 2, 3, 4, and 5. As observed in the figure, the accuracy of classification with API-class is slightly (4%) higher than that of the API-method feature.

5 Conclusion and Future Work

In this paper, we have proposed a technique for classifying Android apps in order to enhance the efficiency of app similarity computation. In our technique, we select API call information as the feature for machine learning and utilize SVM with the RBF kernel for classification. In our experiment, Java classes and methods used by the target Android apps are selected as a possible feature. According to our experimental results, the class-based feature shows better accuracy than the method-based feature for the app classification. Because there are some apps belonging to more than one category at the same time in our dataset, we have classified an app as one category or more than one categories where *k* ranges from 1 to 5. As the *k* increases, the accuracy of classification also increases.

We have begun studies on app classification for efficient software similarity computation. For future work, we plan to select other effective features including permission or string information employed in

Table 1: An abstract view of our dataset

Name	Label(category)	api_1	api_2	...	api_n
app_1	Art & Design	0	0		1
app_2	Finance	0	1		1
...	...	...	...	...	...
app_m	Productivity	1	0		0

Table 2: Some statistics about our dataset

Category	# of app
Art & Design	446
Books & Reference	486
Finance	337
Library & Demo	467
News & Magazines	439
Personalization	327
Photography	301
Productivity	271
Shopping	348
Travel & Local	443
Video Players	449
Weather	410

Table 3: Result of classification with API-class

top-k	Accuracy
1	0.4195
2	0.6053
3	0.7069
4	0.7886
5	0.8503

related works, and use other kernels for SVM and machine learning algorithms such as random forest, neural network, etc.

Acknowledgment

The present research was supported by (1) Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (no. NRF-2015R1D1A1A02061946), and (2) the research fund of Dankook University in 2017.

References

[1] amazon. `https://www.amazon.com/` [Online; Accessed on October 3, 2017].

[2] android developer. `https://developer.android.com` [Online; Accessed on October 3, 2017].

Table 4: Result of classification with API-method

top-k	Accuracy
1	0.3584
2	0.5599
3	0.6609
4	0.7440
5	0.8094

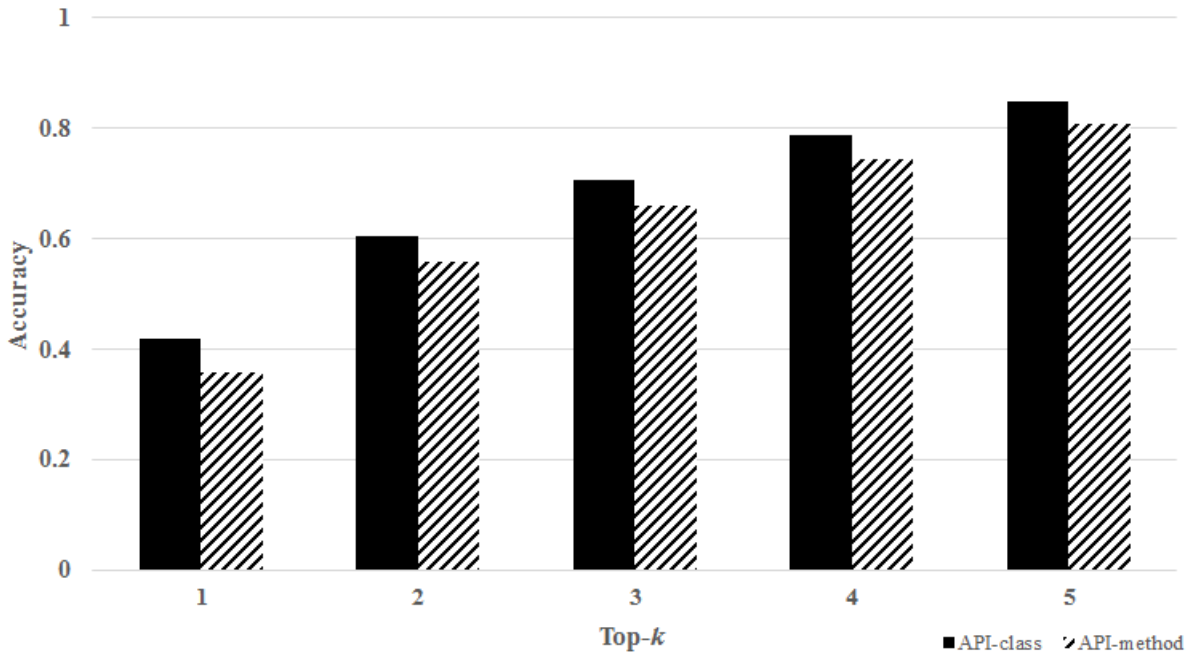

Figure 2: Accuracy of classification depending on top-*k*.

[3] google. `http://play.google.com/apps` [Online; Accessed on October 3, 2017].

[4] J. AH and L. AC. Android app categorization using naïve bayes classifier. *International Journal of Computer Applications*, 122(3), 2015.

[5] A. Desnos. Androguard: Reverse engineering, malware and goodware analysis of android applications... and more (ninja!), 2015. `http://code.google.com/p/androguard` [Online; Accessed on October 3, 2017].

[6] P. Faruki, A. Bharmal, V. Laxmi, V. Ganmoor, M. S. Gaur, M. Conti, and M. Rajarajan. Android security: a survey of issues, malware penetration, and defenses. *IEEE communications surveys & tutorials*, 17(2):998–1022, 2015.

[7] A. Feizollah, N. B. Anuar, R. Salleh, and A. W. A. Wahab. A review on feature selection in mobile malware detection. *Digital Investigation*, 13:22–37, 2015.

[8] S. Hanna, L. Huang, E. Wu, S. Li, C. Chen, and D. Song. Juxtapp: A scalable system for detecting code reuse among android applications. In *Proc. of the 9th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA'12), Heraklion, Crete, Greece, LNCS*, volume 7591, pages 62–81. Springer, Berlin, Heidelberg, July 2012.

[9] H. Huang, S. Zhu, P. Liu, and D. Wu. A framework for evaluating mobile app repackaging detection algorithms. In *Proc. of the 6th International Conference on Trust and Trustworthy Computing (Trust'13), London, UK, LNCS*, volume 7904, pages 169–186. Springer, Berlin, Heidelberg, June 2013.

[10] S. Inc. Number of apps available in leading app stores as of march 2017. `https://www.statista.com/statistics/276623/number-of-apps-available-in-leading-app-stores` [Online; Accessed on October 3, 2017], March 2017.

[11] S. luo and P. Yan. Fake apps feigning legitimacy. *Mobile Threat Research Team published on A Trend Micro Research Paper*, 2014.

[12] B. Sanz, I. Santos, C. Laorden, X. Ugarte-Pedrero, and P. G. Bringas. On the automatic categorisation of android applications. In *Proc. of the 2012 IEEE Consumer Communications and Networking Conference (CCNC'12), Las Vegas, Nevada, USA*, pages 149–153. IEEE, January 2012.

[13] C. Sung-Ha and P. Heewan. An automated classification technique for android application based on software montage. *KIISE transactions on computing practices*, 18(11):756–761, 2012.

[14] C.-Z. Yang and M.-H. Tu. Lacta: An enhanced automatic software categorization on the native code of android applications. In *Proc. of the international multiconference of engineers and computer scientists (IMECS'12), Hong Kong, China*, volume 1, pages 769–773, March 2012.

[15] M. Zheng, M. Sun, and J. C. Lui. Droid analytics: a signature based analytic system to collect, extract, analyze and associate android malware. In *Proc. of the 12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom'13), Melbourne, Victoria, Australia*, pages 163–171. IEEE, July 2013.

Author Biography

Myeonggeon Lee received the B.E. and the M.E. degree in Dept. of Software Science from Dankook University, Korea in 2015 and 2017 respectively. He is currently a Ph.D. student at Dept. of Computer Science and Engineering in Dankook University, Korea. His research interests include computer system security, SCADA system security, mobile security.

Masoud Reyhani Hamedani received the B.E. degree in Dept. of Computer Science from Shahid Bahonar University of Kerman, Iran in 2004, and the M.E. degree in Dept. of Computer Engineering from Payame Noor University, Iran in 2009, the Ph.D. degree in Dept. of Computer Science at Hanyang University, Korea in 2016. He is currently a Postdoctoral researcher in Dept. of Computer Science and Engineering at Dankook University, Korea. His current research interests include knowledge Engineering, enterprise software design and development, web application development, and data mining.

Gyoosik Kim received the B.E in Dept. of Applied Computer Engineering from Dankook Univeristy in 2016. He is currently a master student in Dept. of Computer Science and Engineering at Dankook University, Korea. His research interests include computer system security, mobile security, and software.

Tae Kyung Kim received the B.S. degree from the Department of Applied Computer Engineering, Dankook University, Korea, in 2015 and the M.S. degree from the Department of Computer Scinece and Engineering, Dankook University, Korea in 2017. His research interests include machine learning and computer vision, and their applications.

Seong-je Cho received the B.E., the M.E. and the Ph.D. in Dept. of Computer Engineering from Seoul National University in 1989, 1991 and 1996 respectively. He joined the faculty of Dankook University, Korea in 1997. He was a visiting scholar at Department of EECS, University of California, Irvine, USA in 2001, and at Department of Electrical and Computer Engineering, University of Cincinnati, USA in 2009 respectively. He is a Professor in Department of Computer Science and Engineering (Graduate school) and Department of Software Science (Undergraduate school), Dankook University, Korea. His current research interests include computer security, smartphone security, operating systems, and software protection.