Quantifiable Measurement Scheme for Mobile Code Vulnerability Based on Machine-Learned API Features

Hyunki Kim*, Joonsang Yoo*, Jeong Hyun Yi†
*Department of Software Convergence
Soongsil University, Seoul, 06978, Korea
Email: {hitechnet92,phoibos92}@gmail.com
†School of Software
Soongsil University, Seoul, 06978, Korea
Email: jhyi@ssu.ac.kr

***Abstract*—Owing to open market policies and self-signed certificates, any malicious application developer can easily insert malicious code into Android mobile applications and then distribute them in the Google Play market. Furthermore, even applications that are known to be benign or safe are collecting private information without asking users. Thus, there is a need for a quantifiable measurement scheme that can evaluate the degree of risk posed by an application beyond applications simply being classified as normal or malicious. In this paper, by using ensemble learning, we develop a quantifiable measurement scheme to assess the sensitivity of the Android framework API, and we experimentally evaluate the feasibility of this scheme.**

Keywords–Android; Vulnerability; Application Assessment

I. Introduction

With the increasing number of mobile devices, such as smartphones, tablets, and wearable devices, based on the Android operating system, the use of the Google Play market has also dramatically increased. However, owing to open market policies such as self-signed certificates, the number of users suffering from malicious applications has also greatly increased. For example, a malicious application developer can download an application registered in the market and then re-upload the application with added malignant behavior codes, resulting in many problems such as private information leakage and financial threats [1][2]. These applications can be installed on a users devices and can secretly steal the users location information or encrypt personal information and request money in return for decryption. To implement these behaviors, framework Application Programming Interfaces(API) or user-defined methods are used. User-defined methods also use framework APIs provided by the operating system. Therefore, to understand the behaviors of the application, the APIs used by the application must be analyzed.

In this paper, we generate an API sensitivity ranking by using machine learning with API metadata. API metadata includes the package name, class name, and API name, each of which comprises words that reflect the behavior of the API [3]. Thus, the learning model creates classification rules based on these words, thereby predicting new input data [4]. On the other hand, because the ensemble learning model is more accurate than a single model, we use various learning models in an ensemble to produce a high-accuracy result [5].

This is the first attempt to actually generate API sensitivity ranking. API sensitivity ranking can be used for criteria measuring the risk of an application, thus alert user to potentially risky application. Also, It can make developers refrain from abusing the sensitive API, and therefore spend more attention to secure programming. Previous work has been done to distinguish applications into normal and malicious [6]. However, users might use malicious application because it is distinguished as a normal application. Therefore, applications must be evaluated in a degree of risk to prevent harm.

The remainder of this paper is organized as follows. Section 2 describes the method used to generate the API sensitivity ranking. Section 3 presents the experimental result of generated ranking. Section 4 concludes the paper.

II. Proposed Scheme

The API sensitivity ranking generator, as shown in the following Figure 1, consists of an API Metadata Extractor, Data Preprocessor, and Predictive Model via Ensemble Machine Learning.

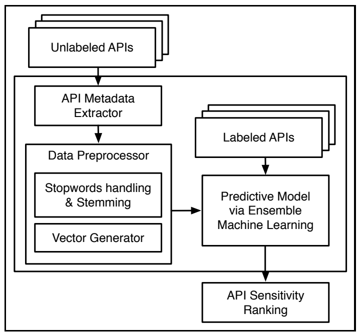

Figure 1. Framework of API sensitivity ranking generator

To generate the API sensitivity ranking, first, we build training data for the nonsensitive and sensitive APIs. To do so, we analyze more than 6,000 malicious applications collected from Contagio [7] and VirusShare [8].

A. API Metadata Extractor

The API Metadata Extractor crawls the metadata of the API in the Android developer site. The API metadata consist of the API name, package name, class name, and API description, which reflects the behavior of the API and the resources that the API accesses. For example, the GetDeviceID method belongs to the TelephonyManager class under the android.telephony package. Through this, it can be seen that GetDeviceID accesses the telephony service. In addition, the

API description Returns the unique device ID of a subscription, for example, the IMEI for GSM and the MEID for CDMA phones explicitly states that it is used to retrieve the device ID, such as IMEI and GSM. When this metadata extraction process is finished, the vocabulary tokenization process is performed for the package name, class name, API name, and API description.

B. Data Preprocessor

1) Stop Words and Stemming: The extracted API Metadata includes stop words such as this, that, and who that, by themselves, do not provide any useful information. Because these words could degrade the performance of the learning model, in this paper, they are removed by defining the following words as stop words [9].

TABLE I. Defined stop words list

```
'i', 'me', 'my', 'myself', 'we', 'our', 'ours',
'ourselves', 'you','your', 'yours', 'yourself',
'yourselves', 'he', 'him', 'his', 'himself',
'she','her', 'hers', 'herself', 'it', 'its',
'itself', 'they', 'them', 'their', 'theirs',
'themselves','what', 'which', 'who', 'whom',
'this', 'that', 'these', 'those', 'am', ...
```

Stemming is a process by which a word is reduced to its word root. It is usually performed by removing any suffixes and prefixes from the word. For example, in the API metadata, some words have the same meaning but in different forms, such as retrieve, retrieved, retrieving, and retrieval. Because these words could also degrade the performance of the learning model, the stemming process proceeds in accordance of the rules [10].

2) Vector Generator: After pre-processing the metadata, they are transformed into a vector that is capable of machine learning. To generate a vector from the metadata, first, a dictionary of all words is created. Then, the metadata is matched with the dictionary.

C. Predictive Model via Ensemble Machine Learning

Ensemble learning is a composite learning model that is constructed by combining various learning models. When new data is given, the individual learning model that makes up the ensemble class votes the class label of the data, and then, the ensemble model collects the votes of the individual learning model and finally predicts the outcome. Under the condition that the classifier outputs are independent, it was proved that the voting combination will always result in a performance improvement compared to a single classifier. The API sensitivity ranking is generated by using the result of the majority voting. As a simple example, assume that there exist five learning models. When new data is entered, if four learning models classify it as sensitive and one learning model classifies it as non-sensitive, then it is given a ranking of 0.8.

III. Experiment

We generate the API sensitivity ranking using ten learning models, in which the highest performance ten learning models were selected using the cross-validation method. The following table shows the API sensitivity ranking created by the ensemble model. The most sensitive APIs are assigned a rank of 1.0 as shown in Table II. For higher ranks, mainly SMS, File, Network, and Contact related APIs are found, whereas for lower ranks, data type, painting, and weather related APIs are found. Our experimental results demonstrate that the top rank results were actually confirmed to be mainly used for malicious applications such as ransomware and Trojans.

TABLE II. Sensitivity API Ranking

Ranking	API List
1.0	URLConnection.getPermission(), TelephonyManager.getDeviceID(), SmsManager.sendDataMessage(), SmsManager.sendTextMessage(), TelephonyManager.getLine1Number(), ...
0.9	LocationProvider.requiresNetwork(), AppWidgetHost.deleteHost(), BasicHttpResponse.getStatusLine(), UserManager.getUserForSerialNumber(), SendSmsResult.getSendStatus(), ...
...	...
0.0	PictureDrawable.getPicture(), DateFormatSymbols.getEras(), TextView.getCompoundDrawablePadding(), Deflater.getAdler(), NumberFormat.getIntegerInstance(), Resources.getColor(), Resources.Theme.getDrawable(), ...

IV. Conclusion

In this paper, we propose a scheme for quantitatively evaluating the risk of an application by generating the sensitivity ranking of the API. Because the API is mainly used to implement the functionality of applications, it is expected that risk assessment using the sensitivity ranking of the API can be more objective compared to conventional risk assessment methods.

Acknowledgment

This research was supported by the Global Research Laboratory (GRL) program through the National Research Foundation of Korea (NRF) funded by the Ministry of Science, ICT, and Future Planning (NRF-2014K1A1A2043029).

References

[1] J. H. Jung, J. Y. Kim, H. C. Lee, and J. H. Yi, “Repackaging attack on android banking applications and its countermeasures,” Wireless Personal Communications, vol. 73, no. 4, 2013, pp. 1421–1437.

[2] A. P. Felt, M. Finifter, E. Chin, S. Hanna, and D. Wagner, “A survey of mobile malware in the wild,” The 1st ACM Workshop on Security and Privacy in Smartphones and Mobile Devices, 2011, pp. 3–14.

[3] Android API Reference, https://developer.android.com/reference/packages.html, Apr. 2015.

[4] P. Domingos, “A few useful things to know about machine learning,” Communications of the ACM, vol. 55, no. 10, 2012, pp. 78–87.

[5] T. G. Dietterich, “Ensemble methods in machine learning,” International Workshop on Multiple Classifier Systems, 2000, pp. 1–15.

[6] M. Lindorfer, M. Neugschwandtner, and C. Platzer, “Marvin: Efficient and comprehensive mobile app classification through static and dynamic analysis,” The 39th IEEE Annual International Computer Software and Applications Conference, 2015, pp. 422–433.

[7] Contagio, http://contagiodump.blogspot.kr/, Apr. 2015.

[8] VirusShare, https://virusshare.com/, Apr. 2015.

[9] H. Saif, M. Fernandez, Y. He, and H. Alani, “On stopwords, filtering and data sparsity for sentiment analysis of twitter,” The 9th International Conference on Language Resources and Evaluation, 2014, pp. 810–817.

[10] D. A. Hull, “Stemming algorithms: A case study for detailed evaluation,” Journal of the American Society for Information Science, vol. 47, no. 1, 1996, pp. 70–84.