A Hybrid Malware Detecting Scheme for Mobile Android Applications

Yu Liu, *Member, IEEE*, Yichi Zhang, Haibin Li, Xu Chen
School of Electronic Information Engineering, Tianjin University, Tianjin 300072, China
liuyu, yichizhang, lihaibin@tju.edu.cn xu.chen.tjutju@gmail.com

***Abstract*—This paper proposes a static-dynamic hybrid malware detecting scheme for Android applications. While the static analysis could be defeated by transformation technique sometimes and dynamic analysis needs a high complexity, the suggested methods can automatically deliver an unknown App to static or dynamic analysis path according to whether the Android App can be decompiled(its feature) which overcomes both weakness. The experimental results show that the suggested scheme is effective as its detection accuracy can achieve to 93.33%∼99.28%.**

I. Introduction

Android based smartphones have been widely used in recent years. Because of the openness of Android system, Android based smartphone are the major target for malwares. Thus, the detection of Android malwares becomes a critical issue in security of smartphones.

In recent years, some malware detecting methods have been proposed, and these methods can be classified into two classes: static analysis and dynamic analysis. Most static analysis methods simply use Permission [1] as feature for analysis [2] [3], which has low complexity. Besides, API [3] is another static analysis feature which can achieve higher accuracy with increase of complexity [4]. Though static analyzing may complete malware detection with low cost, this method cannot be used to process Apps with transformation technique [5]. After testing some existing dynamic monitoring tools, we choose strace [6] to complete dynamic analysis which demands complex computation [7].

As malware detecting will face more challenges such as fast growth of malicious Apps and development of transforming techniques, flexible malware detecting scheme which combines static and dynamic analysis to balance performance and complexity of detecting is desired by practical applications. This paper proposes a hybrid malware detecting scheme which can complete the malware detecting with high accuracy and controlled complexity.

II. The Proposed Malware Detecting Scheme

A. The structure of proposed malware detecting scheme

The proposed scheme is shown in Fig.1. Firstly, The App is decompiled by Apktool [8]. Manifest [9] and Smali [10] files will be created if the App can be decompiled correctly and static analysis can be applied on it. But for Apps which use transformation techniques, the decompiling result may contains nothing useful. To overcome this issue, our scheme employs a static-dynamic hybrid architecture and applies a program which can judge the decompiling result as the automatic switch. The switch can send the App into different analysis process(static or dynamic). Then, trained classifiers such as SVM and kNN using different feature vectors are used to complete the detection for each process.

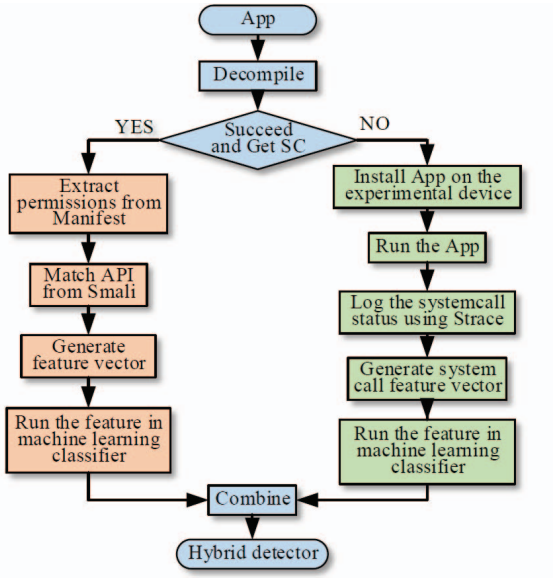

Figure 1. Flow diagram for the hybrid Android malware detection scheme.

B. Static analysis of Hybrid Malware Detecting Scheme

In our hybrid malware detecting scheme, a searching tool is designed to extract the key words of API and Permissions in the App's source code. For Manifest file, the searching tool generates a Permission vector of 151 dimensions. While, for Smali file, it generates an API vector of 3262 dimensions. Two vectors can be joined together to form a feature vector of 3413 dimensions in order to improve the representativeness of the feature.

C. Dynamic analysis of Hybrid Malware Detecting Scheme

In the dynamic analysis process, our scheme generates a feature vector of 345 dimensions. Each element of the vector represents the logged total times of the corresponding system call. The proposal uses the following steps to get the feature of an App:

1. Employ ADB [11] tools to connect the smartphone with experimental platform (such as PC).

2. Use 'ps' command to get the pid (Process Identifier) of the running App.

3. Monitor the App's action according to the pid using Strace.

4. Log the App's system call [12] status for a settled time while the Monkey [13] tool operates the App.

978-1-4673-8364-6/16/$31.00 ©2016 IEEE

5. Deal with the log file and generate a feature vector of 345 dimensions.

Table I
RESULTS OF STATIC ANALYSIS WITH DIFFERENT FEATURE SETS (%)

Algorithm	Permission Feature			API Feature			Permission&API		
	ACC	TPR	FPR	ACC	TPR	FPR	ACC	TPR	FPR
kNN	95.87	97.19	5.46	98.42	97.40	0.57	98.66	97.52	0.20
SVM	96.52	95.68	2.65	99.07	98.48	0.34	99.28	98.68	0.12
Bayes	93.33	94.59	7.94	94.23	99.47	11.01	94.41	99.42	10.6

III. EXPERIMENT RESULTS AND ANALYSIS

The malware dataset is downloaded from Gnome project and benign Apps are downloaded from an App market called Wandoujia [14]. We randomly select 500 samples from both types as our experimental dataset. For each sample in static or dynamic analysis, we extract the corresponding features according to the methods mentioned in Section II.

The performance of our scheme is evaluated by True Positive Rate (TPR), False Positive Rate (FPR) and Accuracy (ACC), which are defined as Eq.(1).

$$\begin{cases} TPR = \dfrac{TP}{TP+FN}, FPR = \dfrac{FP}{FP+TN} \\ ACC = \dfrac{TP+TN}{FP+TN+TP+FN} \end{cases} \quad (1)$$

Here, TP (true positive) represents the number of Android malwares that are correctly detected; FN (false negative) denotes the number of Android malwares that are not detected (predicted as benign application); TN (true negative) is the number of benign applications that are correctly classified; and FP (false positive) is the number of benign applications that are incorrectly detected as Android malware.

To validate the effectiveness of our scheme, different classification techniques are used. These classifiers will produce different classification models for Android Apps and we can pick out the most suitable one through comparing the results.

In Table I, we report the classification results of different classifiers on all three feature sets. The accuracy of three algorithms ranges from about 93% to 99%. The best performance in ACC (99.28%) is achieved by SVM classifier while using combination feature set. Meanwhile, this algorithm has the second highest TPR (98.68%) and lowest FPR (0.12%) which can prove it is the best classifier for Android malware in static analysis. Additionally, Aafer [15] also uses the dataset from Gnome project like us. But his experiment only apply the API as feature and the ACC results are 96.5% (SVM) , 98.9% (kNN) ,97.9% (C4.5) .

Table II
RESULTS OF DYNAMIC ANALYSIS WITH DIFFERENT CLASSIFIER (%)

Algorithms	ACC	TPR	FPR
kNN	87.92	89.40	13.55
SVM	85.75	83.30	11.80
Naive Bayes	90.00	90.85	10.85

The results of dynamic analysis are shown in Table II. In this part of experiments, we extract system call as the feature set and calculate its metrics. The Naive Bayes has the best performance in ACC (90.00%), TPR (90.85%) and FPR (10.85%) so that we can consider it as the best choice in dynamic analysis. Because the complexity of the procedure of extracting system call features is very high, we only use 100 samples in this experiment. So the results are not as good as static analysis, and we will provide further solution for this issue in our future research.

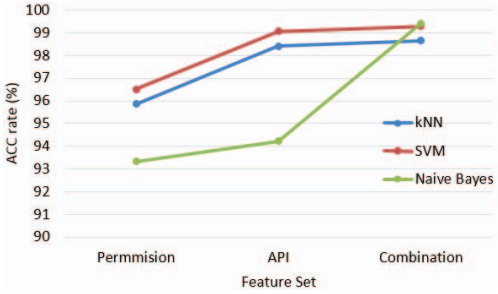

Figure 2. Metrics comparison results between using different classification.

Fig. 2 illustrates the accuracy in malware detection with three different feature sets in static analysis. The API feature performs better than Permission feature and the Combination feature achieved the highest ACC in three cases.The difference is because the Apps don't really use all of declared permissions during execution so that this feature has some distracting information. The API feature is extracted from App's source code and has much more dimensions than Permission, it contains more information and indicates the App's real behaviors. So when our scheme connect both of them together, the new feature achieves a better performance without doubt.

IV. CONCLUSION

This paper proposes a static-dynamic hybrid scheme of Android malware detecting. Whatever the Apps can be decompiled or not, our proposal can automatically apply different detecting methods to malwares, which means it can deal with all the Apps and balance performance and complexity at the same time. Our scheme uses various machine learning methods and different feature sets in detecting. The experiment result shows it can achieve up to 93.33%∼99.28% in accuracy by using SVM classifier and combination feature.

REFERENCES

[1] [online]:"http://developer.android.com/guide/topics/security/permissions.html".
[2] Wang, Wei, et al. "Exploring permission-induced risk in Android applications for malicious application detection.", *IEEE Transactions on Information Forensics and Security* (2014): 1869-1882.
[3] [online]:"http://developer.android.com/reference/packages.html".
[4] Peiravian, Naser, and Xingquan Zhu. "Machine learning for android malware detection using permission and api calls", 2013 IEEE 25th International Conference on. IEEE, 2013.
[5] Rastogi, Vaibhav, Yan Chen, and Xuxian Jiang. "Evaluating Android anti-malware against transformation attacks." *Northwestern University*. March (2013).
[6] [online]:"http://sourceforge.net/p/strace/wiki/Home/".
[7] Burguera, Iker, Urko Zurutuza, and Simin Nadjm-Tehrani."Crowdroid: behavior-based malware detection scheme for android." Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices. ACM, 2011.
[8] [online]:"http://ibotpeaches.github.io/Apktool/".
[9] [online]:"http://developer.android.com/guide/topics/manifest/manifest-intro.html"
[10] [online]:"https://code.google.com/p/smali/".
[11] [online]:"http://developer.android.com/tools/help/adb.html".
[12] [online]:"https://en.wikipedia.org/".p
[13] [online]:"https://developer.android.com/tools/help/monkey.html".
[14] [online]:"https://www.wandoujia.com/".
[15] Aafer, Yousra, Wenliang Du, and Heng Yin. "DroidAPIMiner: Mining API-level features for robust malware detection in android." Security and Privacy in Communication Networks. Springer International Publishing, 2013. 86-103.

978-1-4673-8364-6/16/$31.00 ©2016 IEEE