Malware Detection in Android Applications Using Integrated Static Features

A. S. Ajeena Beegom(✉) and Gayatri Ashok

Department of Computer Science and Engineering,
College of Engineering Trivandrum, Thiruvananthapuram, Kerala, India
ajeena@cet.ac.in, gayu.lalitha@gmail.com

Abstract. Android operating systems based mobile phones are common in nowadays due to its ease of use and openness. Hundreds of Android based mobile applications are uploaded in the internet every day, which can be benign or malicious. The increase in the growth of malicious Android applications is alarming. Hence advanced solutions for the detection of malware is needed. In this paper, a novel malware detection framework is proposed that uses integrated static features and Support Vector Machine (SVM) classifier. The static features considered include permissions, API calls and opcodes. Out of these features, most significant ones are selected using Pearson correlation coefficient and N-grams. Each of these features are then integrated and fed to a classifier. The experimental evaluation of the proposed method and comparison with existing methods shows that the proposed framework is better.

Keywords: Android · Malware detection · Classification · Static features

1 Introduction

Android based devices have become famous in this digital era spanning over millions of users. This makes it the major target of attack through malicious application programs known as malware. The malware available in the market are *Botnets*, *Rootkits*, *SMS Trojans*, *Spyware*, *Installer*, *Ransomware*, *Trojans*, etc.

Botnet is a collection of devices known as *bots* that are connected through internet. Attacks such as distributed denial of service (DDoS) is performed by *Botnets*. *Rootkit* is a group of computer software structured to enforce access to a computer system that is restricted otherwise. *SMS Trojans* send SMS messages stealthily to premium numbers and without the user consent. *Spyware* is a software which is capable of gathering information regarding a person without their consent and communicate those information to another entity without the consumer knowing it. *Ransomware* is a malicious software that demands an amount as ransom for access to user data or it threatens to either publish his

S. M. Thampi et al. (Eds.): SSCC 2019, CCIS 1208, pp. 1–10, 2020.
https://doi.org/10.1007/978-981-15-4825-3_1

data or block his data if ransom is not paid. *Trojans* are capable of modifying or deleting data from the device without the user consent infecting personal computers when the device is connected via USB port.

Figure 1 shows the top 10 malware sent out by the MS-ISAC Security Operations Center (SOC) (https://www.cisecurity.org/ms-isac/) showing their percentage shares in January 2018. Each of them belong to any one of the malwares described above. This has motivated us to investigate more on the topic of malware detection techniques for Android applications. In this work, permissions, API calls and source code are analyzed and Android applications are classified as benign or malicious using machine learning techniques.

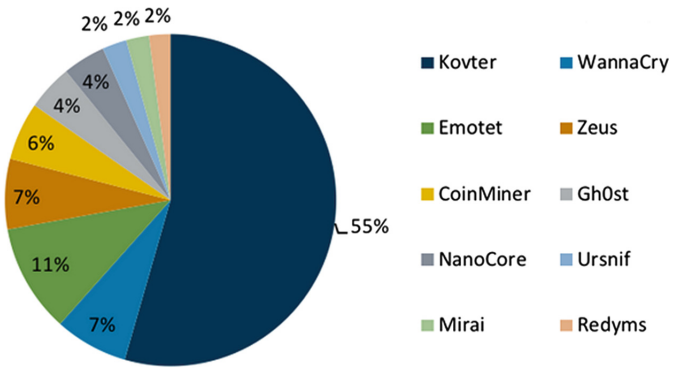

Fig. 1. Malware Seen in January 2018 (https://www.cisecurity.org/ms-isac/)

2 Related Works

Among the static-based approaches for malware detection using permissions, the technique used by Ju [1] and Pehlivan et al. [2] are simple, but corresponds to a bulk feature set that is difficult to process. Wang et al. [3] use three different ranking techniques for identifying the risk-induced permissions from feature set to reduce processing time. Li et al. [4] propose a three-step procedure to identify significant permissions. The steps include permission ranking with negative rate, support based permission ranking and permission mining with association rules. A classifier is then used to classify the application as malicious or benign. Aung et al. [5] suggest a three steps procedure consisting of feature selection, K-means clustering model generation and classification. Kang et al. [6] uses creators information along with permissions, API sequences and system commands for malware detection and classification. SVM based classification is used by Li et al. [7] that uses risky permissions and vulnerable API calls. Machine learning based approaches and multiple classifiers are employed by Milosevic et al. [8] for static analysis.

DroidAPIMiner [9] considers the critical API's capable of distinguishing between malicious and benign applications. Another API based analysis tool is DroidAnalyzer [10] for applications where it decompresses the *apk* files and parses through the assembly codes to come up with keywords and risky API's related to abnormal behavior by assigning suspicion level to each application. Since it uses MD5 hash values in the database, overhead of maintaining a bulky database is avoided, but it is unable to detect code obfuscation. Atici et al. [11] proposed a static feature based technique using control flow graphs. This method is efficient for malware using obfuscation techniques, but it is weak against zero-day attacks. Zhu et al. [12] have proposed a complex method based on deep learning and static analysis that use APIs and the source code of Android applications. Obfuscation-resilient malware detection is proposed in [13].

3 Proposed Approach

The proposed system works in four different phases, namely disassembling, feature extraction, feature selection and classification. Three static features are combined to give the maximum accuracy using appropriate selection techniques. The proposed system architecture is shown in Fig. 2.

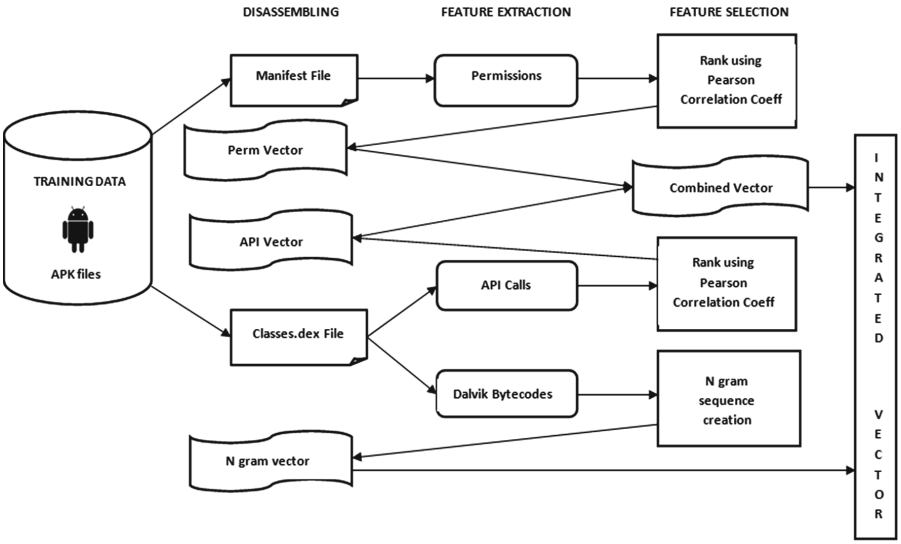

Fig. 2. Proposed framework using integrated static features

3.1 Disassembling

Android Applications contain many prominent files and folders, namely, *META-INF*, *lib/*, *res/*, *Android Manifest.xml*, *classes.dex* and *resources.arsc*. Among

these, the *manifest.xml* and *classes.dex* has the permissions and API calls used by the application respectively. To classify an application as malicious or benign, the Android application package (the *apk* file) needs to be disassembled. This is done using tools such as *apktol*, *dex2jar*, *jdgui* etc. Features such as permissions and API calls are extracted from these which is used for building the feature set.

3.2 Feature Extraction

Permissions: Permissions are the basic access rights given to an Android application that helps to control or enforce security against the misuse of sensitive information of users. These permissions can be extracted from the Android *manifest* file from the Android package. Among the permissions considered, significant permissions can be extracted using various techniques. Python libraries are used to extract permissions from the manifest file and to eliminate duplicates.

API Calls: These are a set of procedures and protocols for building an Android application. This information is extracted from the source code and considered as one of the key features. Many malware detection systems are control flow graph (CFG) based or signature-based which uses APIs as features. CFGs are used to understand the flow of control between the calls in order to study behavioral patterns and to detect any suspicious activity. The signature-based approach can also be implemented which searches for API's within the bytecode against a list of critical API's, suspicious keywords or a combination of both.

Androguard is an open source tool that can be used to manipulate the *apk* files to extract various information. *Analyze APK* option from *Androguard* is used to extract the calls from the external classes.

Dalvik Bytecode: An Android application package or an *apk* file is a zip file which is a set of many other files, namely the manifest file, dex or Dalvik executables and resource files. The bytecode interpreted by the Dalvik Virtual Machine is called *DEX code* (Dalvik EXecutable code). The *DEX code* can be obtained by converting Java bytecode using the dex tool. Human-readable Dalvik bytecode are contained within methods.

3.3 Feature Selection

Pearson Correlation Coefficient: A row vector is constructed for each application corresponding to the permissions and API calls that the application uses. Therefore, a matrix is created for the considered applications appending them with their corresponding class variables as a final column. The class variable signifies to which class the application falls to. A value of 1 is given for a benign class and 0 is given for a malicious class. The pre-processing begins by performing ranking over this constructed matrix using Pearson correlation coefficient [1]. The permission or API variable obtained after disassembling is denoted by X, which is the column vectors of the constructed matrix and the class variable is denoted by C. Column vectors can have a value of 1 which specifies that the

corresponding permission is used or 0 specifies that it is not used. The equation (1) can be used to compute the relevance of these variables, using covariance and variance over given data.

$$R(X,C) = \frac{cov(X,C)}{\sqrt{var(X)var(C)}} \tag{1}$$

where *cov* represents the covariance between two sets of data and *var* represents the variance over a set of data. Since our case deals with binary classes and Boolean variables, equation (1) is changed to equation (2).

$$R(X,C) = \frac{\sum_{n=1}^{N}(X_n - \overline{X}) * (C_n - \overline{C})}{\sqrt{\sum_{n=1}^{N}(X_n - \overline{X})^2 \sum_{n=1}^{N}(C_n - \overline{C})^2}} \tag{2}$$

Table 1. The 10 most risky permissions and API calls obtained

Rank	Score	Permission	Score	API Calls
1	0.422	READ_EXTERNAL_STORAGE	0.432	abortBroadcast()
2	0.385	RECIEVE_SMS	0.426	chmod()
3	0.380	BIND_GET_INSTALL_PACKAGE	0.401	startService()
4	0.204	BLUETOOTH	0.394	writeTextMessage()
5	0.179	GET_PACKAGE_SIZE	0.382	getrunningTask()
6	0.166	WAKE_LOCK	0.343	sendTextMessage()
7	0.164	CSD_MESSAGE	0.302	setupWindow()
8	0.162	PACKAGE_USAGE_STATS	0.278	setInputstream()
9	0.160	INSTALL_SHORTCUT	0.256	startActivity()
10	0.159	EXPAND_STATUS_BAR	0.224	setVisibility()

The average of all sample values of X is denoted by $\overline{X}$, X_n denotes the total number of samples considered and n can have a value ranging from 1 to N. $R(X,C)$ has a value in the range [−1,1], and if it holds a value 0 then it indicates that X and C are independent, whereas a value of 1 indicates that there is a strong positive correlation between X and C and a value of -1 indicates a strong negative correlation between the variables. In this work we assumed that $R(X,C) = 1$ means that the permission request of X makes applications highly risky whereas $R(X,C) = -1$ means that the permission request of X makes applications less risky. This ranking method is used to consider only the top k features for classification avoiding unnecessary processing. Table 1 shows the

scores ($R(X, C)$ value) of the top ten ranked permissions and API calls obtained on our evaluation. Pearson correlation coefficient is used for feature selection over the extracted permission and API vectors.

N-gram: N-gram is a continuous sequence of N items which is extracted from a text sample or a piece of speech. The output of the N-gram opcode extraction is a vector of unique N-gram opcodes from all the classes of the application containing the frequency of each unique N-gram opcode. The opcode sequence generation is shown in Fig. 3.

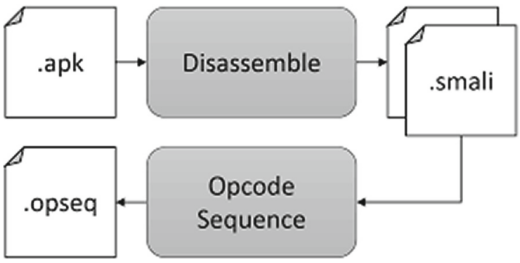

Fig. 3. N-gram opcode generation

An Android application package or *apk* file is a zip file which is a set of many other files, namely, the manifest file, dex or Dalvik Executable and resource files. Disassembling of dex files is done by a process called baksmaling. Set of smali files is extracted from the dex files and each class is represented using a smali file and all the methods are contained within the class. Using N-gram as the feature selection technique a N-gram vector is generated.

3.4 Classification

As shown in Fig. 2, the combined vector of permissions and API calls is integrated with the N-gram vector to give the integrated vector which is then fed to the classifier. The classifier then classifies the test data samples into benign or malicious. Support Vector Machines (SVM), Random Forest or Decision Trees can be used for classification purpose. Depending on the dataset, a cross fold validation is applied for obtaining the training and the test data sets and a corresponding classifier that yields the best results can be selected.

4 Experimental Setup and Evaluation

The analysis is conducted on 500 malicious and 500 benign *apk* samples collected from *Virus share* and *Google Play Store* respectively. The environment is set up on a laptop that runs on *Ubuntu 16.04* and the analysis was conducted using

Python 3.6. Permissions are extracted using the *aapt dump* command provided by *Python* which writes these into text files which is further used for classification. Ranking of permissions are done using Pearson Correlation Coefficient and the top 220 permissions from the extracted 450 are considered for classification. To identify the best classifier, the top 220 permissions identified by the ranking method using Pearson Correlation Coefficient are fed to SVM classifier, KNN classifier and Random Forest classifier by varying the number of top permissions. Figure 4 shows the variation in accuracy levels by each of these classifiers according to the change in the number of top-ranked permissions as features. As seen in this figure, the SVM based classifier gives better results than the other two classifiers for any number of permissions as features.

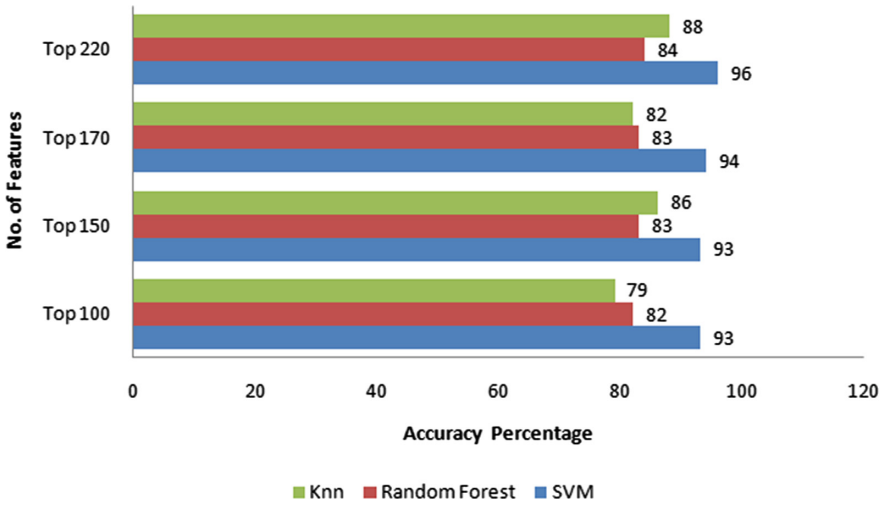

Fig. 4. Classification accuracy versus number of permissions

Androguard is used to extract API calls from the *apk* files and permissions are extracted using Python tools. Baksmaling is executed in order to obtain the N-gram sequences from the sample data. Individually these features are again fed to SVM classifier and Random Forest classifier and the results are analyzed. Table 2 shows the classification results of individual as well as the integrated approach using both SVM and Random Forest classifiers which yield the maximum accuracy using cross validation. The total number of permissions taken for study include 220 with the number of API calls as 460 and the N-grams as 400. The proposed framework achieves a precision of 0.91 and a recall of 0.90 with SVM classifier.

Table 2. Classification results

	Classification accuracy	
Features	SVM	Random forest
Using API calls alone	90%	83%
Using permissions alone	93%	84%
Using Dalvik bytecodes Alone	90%	84%
Integrated static approach	96%	84%

As the SVM based classifier gives better results than Random Forest based classifier, the proposed integrated static approach is again analyzed experimentally. The true positive rate (TPR) is a quality measure that is used in machine learning to measure the proportion of actual positives that are correctly identified. The variation of TPR on malware detection with the percentage of features taken for classification using SVM classifier is shown in Fig. 5.

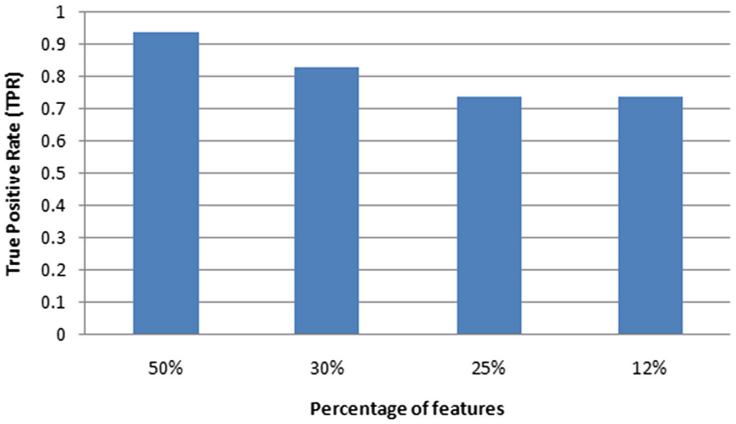

Fig. 5. TPR versus percentage of features

Table 3 shows the comparison of the proposed SVM based integrated static approach with existing systems. SigPID [14] is an Android malware detection system that applies a ranking technique over permissions and does a multi-level pruning process to extract only the significant ones that attains an accuracy of 91%. There are other systems that use sensitive APIs as features [15] which secures an accuracy percentage of 92%. The comparison is also done for existing SVM based systems that uses incremental SVM for classification [16] that has an accuracy of 90.5% whereas the proposed system has an accuracy of 96%.

These studies shows that the proposed framework of integrated static features using SVM classifier for the detection of Android applications as malicious or benign is better than the existing algorithms.

Table 3. Comparison with existing systems

System	Accuracy
Sigpid [4]	91.97%
Detection based on sensitive APIs [15]	92%
Detection based on multi-modal features [17]	94%
Detection based on incremental SVM [16]	90.5%
Proposed integrated system using SVM classifier	**96%**

5 Conclusion and Future Work

Different malware families and their attacks on the current scenario are discussed. To overcome these attacks, an integrated static approach for Android malware detection is proposed. The approach uses various static features in a way to yield the maximum accuracy. From the experimental evaluation and analysis, it is evident that the support vector machine is the best classifier to work with the sample data chosen. The integrated approach attains the maximum accuracy in comparison to the individual static-based methods yielding an accuracy of 96% for the test data. To continue with the work, more dynamic features can be added to enhance the classification efficiency.

References

1. Ju, X.: Android malware detection through permission and package. In: Proceedings of International Conference on Wavelet Analysis and Pattern Recognition, vol. 1, p. 1. IEEE (2014)
2. Pehlivan, U., Baltaci, N., Acartürk, C., Baykal, N.: The analysis of feature selection methods and classification algorithms in permission based android malware detection. In: Proceedings of IEEE Symposium on Computational Intelligence in Cyber Security (CICS), pp. 1–8 (2014)
3. Wang, W., Wang, X., Feng, D., Liu, J., Han, Z., Zhang, X.: Exploring permission-induced risk in android applications for malicious application detection. IEEE Trans. Inf. Forensics Secur. **9**(11), 1869–1882 (2014)
4. Li, J., Sun, L., Yan, Q., Li, Z., Srisa-an, W., Ye, H.: Significant permission identification for machine learning based android malware detection. IEEE Trans. Industr. Inf. **14**(7), 3216–3225 (2018)
5. Aung, Z., Zaw, W.: Permission based android malware detection. Int. J. Sci. Technol. Res. **2**(3), 228–234 (2013)
6. Kang, H., Jang, J., Mohaisen, A., Kim, H.K.: Detecting and classifying android malware using static analysis along with creator information. Int. J. Distrib. Sens. Netw. **11**(6), 479174 (2015)
7. Li, W., Ge, J., Dai, G.: Detecting malware for android platform: an SVM-based approach. In: Proceedings of 2nd IEEE International Conference on Cyber Security and Cloud Computing, pp. 464–469 (2015)
8. Milosevic, N., Dehghantanha, A., Choo, K.R.: Machine learning aided android malware classification. Comput. Electr. Eng. **61**, 266–274 (2017)

9. Aafer, Y., Du, W., Yin, H.: DroidAPIMiner: mining API-Level features for robust malware detection in android. In: Zia, T., Zomaya, A., Varadharajan, V., Mao, M. (eds.) SecureComm 2013. LNICST, vol. 127, pp. 86–103. Springer, Cham (2013). https://doi.org/10.1007/978-3-319-04283-1_6
10. Seo, S.H., Gupta, A., Sallam, A.M., Bertino, E., Yim, K.: Detecting mobile malware threats to homeland security through static analysis. J. Netw. Comput. Appl. **38**, 43–53 (2014)
11. Atici, M.A., Sagiroglu, S., Dogru, I.A.: Android malware analysis approach based on control flow graphs and machine learning algorithms. In: Proceedings of 4th International Symposium on Digital Forensics and security (ISDFS), pp. 26–31. IEEE (2016)
12. Zhu, R., Li, C., Niu, D., Zhang, H., Ki-nawi, H.: Android Malware Detection Using Large-scale Network Representation Learning, p. 1. Cornell University (2018)
13. Suarez-Tangil, G., Dash, D.K., Ahmadi, M., Kinder, J., Giacinto, G., Cavallaro, L.: DroidSieve: fast and accurate classification of obfuscated android malware. In: Proceedings of Seventh ACM on Conference on Data and Application Security and Privacy, pp. 309–320 (2017)
14. Sun, L., Li, Z., Yan, Q., Srisa-an, W., Pan, Y.: SigPID: significant permission identification for android malware detection. In: Proceedings of 11th International Conference on Malicious and Unwanted Software (MALWARE), pp. 1–8. IEEE (2016)
15. Narayanan, A., Chandramohan, M., Chen, L., Liu, Y.: Context-aware, adaptive, and scalable android malware detection through online learning. IEEE Trans. Emerg. Top. Comput. Intell. **1**(3), 157–1575 (2017)
16. Li, Y., Ma, Y., Chen, M., Dai, Z.: A detecting method for malicious mobile application based on incremental SVM. In: Proceedings of 3rd IEEE International Conference on Computer and Communications (ICCC), pp. 1246–1250. IEEE (2017)
17. Ban, T., Takahashi, T., Guo, S., Inoue, D., Nakao, K.: Integration of multi-modal features for android malware detection using linear SVM. In: Proceedings of 11th Asia Joint Conference on Information Security (AsiaJCIS), pp. 141–146. IEEE (2016)