Proposing a New Feature for Structure-Aware Analysis of Android Malwares

Shahrooz Pooryousef
Department of Computer Engineering
Data and Network Security Lab
Sharif University of Technology
Tehran, Iran
pooryousef@ce.sharif.edu

Kazim Fouladi
Department of Computer Engineering
Faculty of Engineering, College of Farabi
University of Tehran
Tehran, Iran
kfouladi@ut.ac.ir

***Abstract*—Android is a major target of attackers for malicious purposes due to its popularity. Despite obvious malicious functionality of Android malware, its analysis is a challenging task. Extracting and using features that discriminate malicious and benign behaviors in applications is essential for malware classification in using machine learning methods. In this paper, we propose a new feature in Android malware classification process which in combination with other proposed features, can discriminate malicious and benign behaviors with a good accuracy. Using components such as activities and services in Android applications' source code will lead to different flows on invoking between application's components. We consider this flows of invoking between as a new feature which based on Android malware behaviors analysis, is different in benign and malicious applications. Even tough inter-app communications have been covered in many researches, using intra-app communication as a feature in Android malware analysis field using ML methods has been seldom addressed. Our results show that we are able to achieve an accuracy as high as 85% and a false positive rate as low as 10% using SVM classifier on a data-set contain 10,320 Android malware and benign applications.**

I. Introduction

Nowdays, smart phones prepare common functionality of traditional computers for users because of the improvements in their hardware and computing capabilities [1]. There are so many third party markets which contain applications in different categories for users [2]. However, the trustworthiness of third party applications' developers can not be verified and these markets usually are the main origin of malicious applications distribution [3], [4]. Based on McAfee reports in 2013, there are 68,000 malicious Android applications which indicate a 197% increase over 2012 [2]. These malwares steal users private data, sending premium-rate SMS messages, and generally do malicious actions on users devices.

Using Machine Learning (ML) in classification algorithms for malware analysis [5], [6], [7] has been considered in many researches to enable better detection of unknown malwares. While malware classifiers with ML algorithms have the potential to detect Android malwares on market shares, there are a number of challenges in determining which features are the most effective in malware detection. In addition to use a good classifier and also a good data-set in classification process, the most important concept in application analysis is providing a good set of features in which illustrates the structure and discriminates malicious and benign behaviors obviously. Furthermore, a problem was raised when we want to decide how many features we would choose for the classification task from the ranked lists in the output of a feature selection algorithm.

In order to have a good sense about application structure in the application analysis process, in this paper, we have proposed a new feature for structure-aware analysis of Android malwares. Android applications can use different components in their program codes based on different attributes of functionalities [8]; For example, Services are used for background processing and do not have any interface for interacting with users. Using these components in application source code will lead to different flows on invoking application's components in benign and malicious applications which have been seldom addressed in Android malware analysis field. We have used this information alongside of other proposed features in previous works in an SVM classifier. The proposed solutions in this research area have focused on inter-app communication analysis [9], [8], [10], [8] or finding vulnerability in application interface via intra-app communication analysis [11], [12], [13]. Our approach is based on intra-app communication analysis for malware detection usin ML classifiers. In order to find the optimum combination of features with different sample data-set, we use, six sets with different sample sizes for each combination of component-flow with other features. Our results show that our proposed feature could help us to find malicious applications in a data-set containing 10,341 benign and malicious applications with 84% accuracy and little false positive (10%) samples.

The rest of this paper is organized as follows: we introduce our proposed feature and our system design for detecting Android malwares in Section II, and present a detailed evaluation in Section III. Related works are discussed in Section IV, and Section V concludes the paper and enumerates limitations of our approach.

II. Proposed Feature and Motivation

A. Over View

In this section, we will explain the structure and steps of our ML-based classifier, features which we will use in our classification process, and our methodology for extracting these features from "apk" files. Furthermore, we will explain our motivation for using a new feature in classification process. In Fig. 1, two phases of training and detection phase of our SVM classifier have been shown. In the training phase, we first, disassemble applications' apk files, extract different features and information from generated Smali files of the applications, and generate SVM's support vectors of features. Working directly on Smali codes [14] helps us to overcome the limitations that obfuscation techniques create for most static analysis tools and make them inefficient. For extracting some of our used features, we have used Androguard tool set. Androguard [15] is a tool set that provides a fairly easy-to-use interface for analyzing and reverse engineering Android applications at byte code level.

Afterwards, we use extracted features to train a single-class Support Vector Machine (SVM) [16], using the Scikit-learn framework [17], which provides a firendly interface to LIBSVM [18]. SVM is a supervised learning method that proceeds through dividing the training data by an optimal separating hyperplane. Then, we divide our data-set for training and detection phase and train our classifier with different couple of feature sets.

B. Motivation and Extracted Features

In order to use any classification algorithm, we need to first extract appropriate features from application in a way that extracted features exhibit the application structure and behaviors. Android malwares use different trikes to bypass existing detection methods [19],[20] and [21]. One of these techniques is using different kinds of components that provide this ability for malicious application to run its malicious actions in the background or in a specific time [19].

Android applications can use four different components in order to perform their actions. These components are activity, service, broadcast receiver, and content provider. Activities are applications' interfaces. Services are used for background processing. Anytime that an action needs to perform a long time operation in which do not need user interactions, developers use this component. Broadcast receivers as an interesting class in Android programming, are triggered via broadcast messages in the system or received messages from other components of applications. Using content providers, developers can store and retrieve data from SQL Lite data base [20]. Android uses Intents for provide a mechanism for interacting an application's components with each other. An Intent object, is a passive data structure that has an abstract description of an operation to be performed [10].

Using components such as broadcast receivers is a very common behavior in Android malwares [20], [21]. *Different component usage in the application's source code can generate flows on invoking between different components which is discriminate in benign and malicious applications.* An instance of this flow between Android components has been addressed in Fig. 2. Based on Fig. 2, receiving an event inside the system, for example, can trigger a broadcast receiver component, and then, an activity is invoked by this component and a method from a service component will be triggered afterwards.

In below, we have explained our proposed feature and three of the most used features in previous researches which have been used in combination with our proposed new feature in evaluation section.

1) ***Component Invoking Flow:*** In training and detection of our proposed system, for every given application, we extract its components' name from the manifest file. Each Android application has a XML file, called "Manifest.xml", which contains some important information about the application such as names of its components, and required permissions of the application. For each component, we identify its methods and afterwards, we construct a **C**omponent **F**low **G**raph (**CFG**) between its components. **CFG** is an abstract representation of a flow in which vertices represent the name of application's components, and edges represent the possible paths of method invoking of different components.

2) ***Permissions:*** Every application in Android system must define different permissions in order to access certain resources of the Android operating system. For example, the permission "android.permission.INTERNET" requests the right to access the Internet, and "android.permission.READ_CONTACTS" requests the right to access contacts database on the user's device [22]. These permissions have been used in many researches for malware analysis purpose [21], [3], [23].

3) ***Security Sensitive APIs:*** Another feature which has been used in [21] and [22] as a good feature for malware analysis is **S**ecurity **S**ensitive **API**s (**SS-API**s) in the application source code. **SS-API**s are those APIs in the Android system which handle a security sensitive action in the system (for example, access to the camera) [19]. These APIs trigger Android access control mechanism whenever they are invoked. We have used Felt et al. [22] research results for identifying **SS-API**s inside the byte codes of applications.

4) ***Application Intent Information:*** There is different information about components of an application such as intent filters inside the manifest file. For example, an intent filter is an expression that specifies type of intents which an specific component in the application would like to receive. Whenever a new event occurs in the system, such as receiving a new sms, Android system will find the appropriate component for invoking based on the contents of the intent filters declared in the manifest file. Malicious applications can capture system events with this capability and trigger their malicious codes in specific time or conditions. We have used these information as a feature alongside other features.

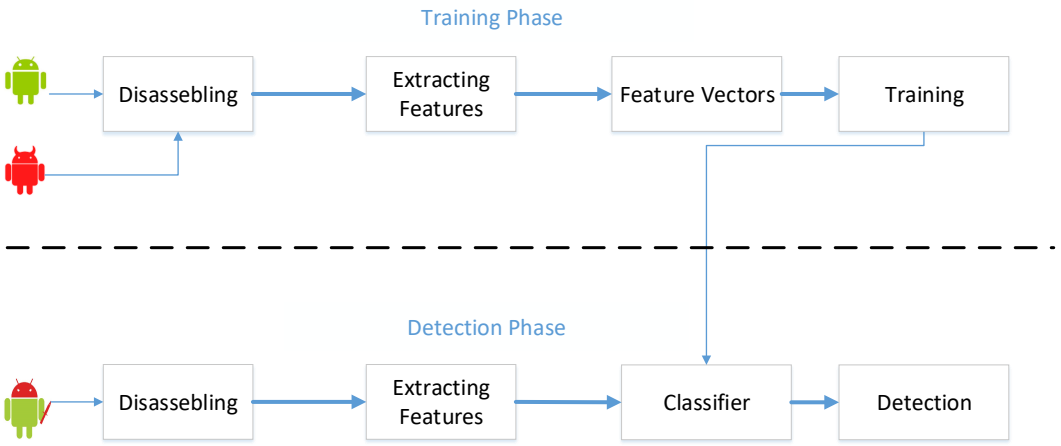


Fig. 1: The steps of our static analysis framework

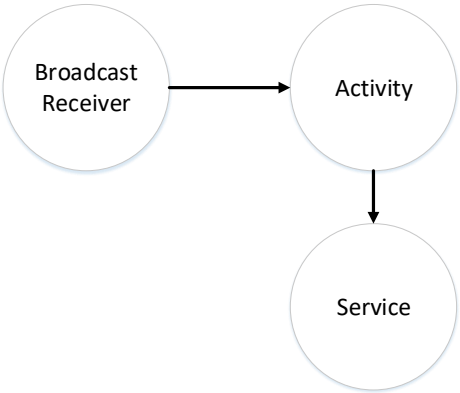


Fig. 2: An instance of flow of invoking between different components of an application

III. RESULTS

Our method is evaluated on a large data-set of real Android malicious and benign applications. In the following, we begin by providing a detailed description of our data-set in Section III-A, then we proceed to evaluate our classifier's ability to detect malicious Android applications in Section III-B.

A. Data Set

Our data-set consists of 5,102 benign applications downloaded from official Google play store that we have collected in July 2016 and 5,287 malicious Android applications obtained from VirusShare and ISBX repositories which are publicly available for researchers. Among these malwares, 4,067 Android malwares have been downloaded from VirusShare repository [24] and 1,220 malwares from ISBX center [25].

B. Experiments and Results

Our goal is to find the impact of using our proposed feature besides of other features used in previous researches to find malicious application in a data-set contain malware and benign applications. To evaluate our SVM classification model using different features, we have used split validation. That is, we randomly divide our data-set into different instances for training and testing (detection) phases. We first, train the classification model using training set (2/3) and then, use the testing instances (1/3) to evaluate our SVM classifier using different metrics. In evaluation, we have used below criterions.

- **True Positive (TP)**: The number of malware instances correctly identified.
- **True Positive Rate (TPR)**: The proportion of malware instances that were correctly classified.
- **True Negative (TN)**: The number of benign instances correctly identified.
- **True Negative Rate (TNR)**: The proportion of benign instances that were correctly classified.

The results of TP and TN, using different features separately in our classifier, have been shown in Fig. 3. Note that, increasing the sample size of benign and malicious applications, causes TP and TN to increase which the best increase is for component invoking flow (indicated as Components flow in Fig. 3) and permission features. Even though TP and TN rate in using flow of invoking between different components and permission features (3a and 3b) increase as the sample size is increased, it does not lead to a very good accuracy. So, for achieving better TP and TN rate, we ran another experiment, using our same data-set for training and testing, in order to find the best combination of two features which will produce the best performance in accuracy, TPR, and TNR. Even though using **SS-API** and component invoking flow features separately generate good TP and TN values in malware detection, unexpectedly, as shown in Fig. 4a and 4b, the classifier's TN and TP rate is not the optimum. The optimum combination is for using permission and component flow set with 84% TP rate. The reason is that using features such as permission can not illustrate a good sense of application's structure lonely.

We checked some of our classifier's labeled applications manually, and found that some of our samples of benign and malicious applications have the same set of permissions. In addition, there is a little similarity between the number of same flow between different components in malicious and benign applications.

For the overall performance, we used the accuracy (the total number of benign and malware instances correctly classified divided by the total number of the dataset instances), TPR, and TRN as has been depicted in Fig. 4. These metrics were calculated as below:

$$TPR = \frac{TP}{TP + FN}$$
$$TNR = \frac{TN}{TN + FP}$$
$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

Though the results obtained by other methods via other classifiers maybe are a little better than ours, we can identify from our experiments that our proposed feature can be used in Android malware analysis as a potential feature. Due to limited space of this paper and as our main objective is to find the influence of using our proposed feature in Android malware classification, we have not considered different classification algorithms in our evaluation and we let this analysis as a future work.

IV. Related Work

In the area of Android inter-app comminication or ICC analysis a broad body of work has been published in which we can classify them in inter-app and intra-app ICC analysis. There are various methods for detecting Intent hijacking or spoofing and ICC vulnerability via tracking sensitive information within an app [26], [27], [28]. CHEX [9] is a static analysis method to automatically vet Android apps for modeling component hijacking vulnerabilities from a data-flow analysis perspective. ComDroid [8] investigates the vulnerabilities related to ICC. Epicc [27] is a sound static analysis technique for ICC specifications and vulnerabilities. DroidSafe [10] is a static information flow analysis tool that analyzes potential leaks of sensitive information in real-world Android applications by tracking sources and sinks APIs.

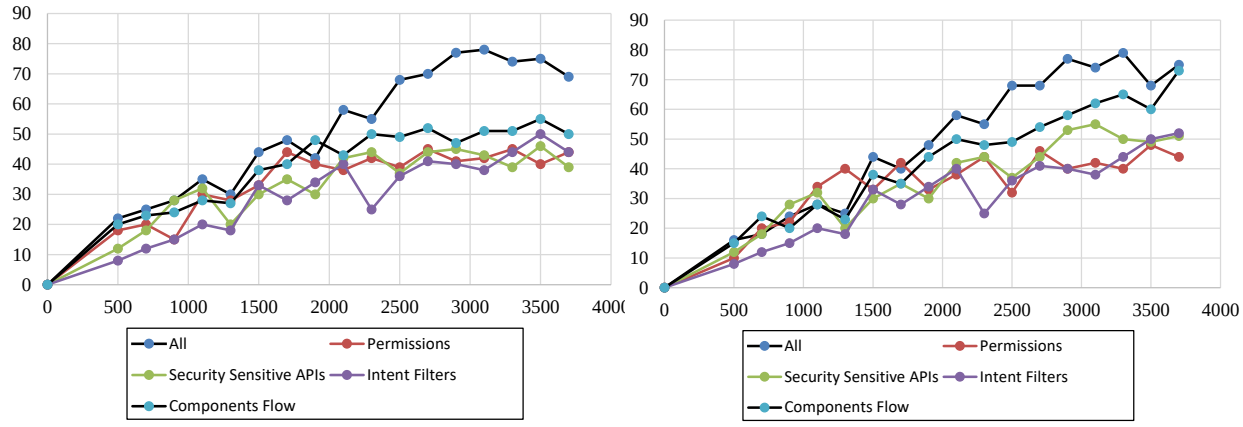
For related inter-app ICC analysis, MR-Droid [11] uses MapReduce framework for highly scalable and accurate inter-app ICC risk analysis in a large data set of Android applications. ICCDetector [12] uses applications' ICC patterns for detecting those "advanced malwares" which exploit inter-application collaboration and are not detectable based on their resources. In contrast to our approach in which we just consider ICC patterns extracted from application intra-app ICC, the main goal of ICCDetector is to examin the communication between applications and Android system. IccTA [13] and [29] combine maltiple applications and analyse information flow in the generated single app. Some other approaches such as IntentFuzzer [30], INTENTDROID [31], and FineDroid [32] perform dynamic testing and information tracking in Android application for privacy monitoring and detecting unsafe handling of ICC interfaces. DroidMiner [33] and DroidAPIMiner [34] scan application's apk file and extract sensitive API calls for detecting malwares using classification algorithms.

Android malware analysis also has been covered via different techniques such as machine learning and data mining [23], [35] and [36]. A good survey on this field can be found in [5]. In static analysis, static features which are extracted from applications apk files have been used in classifiers and applied to find new patterns in application analysis. Dendroid [37] uses text mining approach to analyze application's code and classify malware families. DROIDRANGER [3] implements a combination of a permission-based behavioral foot printing scheme to detect samples of already known malware families and a heuristic-based filtering scheme to detect unknown malicious applications. DroidMat [38] uses different extracted features such as permission and security sensitive APIs from application apk files and applies different clustering algorithms for malware detection. [20] uses Smali code instrumentation in Android application in order to trigger applications different components automatically for malware detection.

Behind malware analysis, there is a broad body of work on tracking users privacy and proposing new and fine-grained access control in Android. TaintDroid [39] uses data tainting for tracking data flows inside the applications for preserving user privacy. Different new access control at operating system [37], browser and operting system level [40] and application level [41] have been proposed for restricting untrusted applications functionalities. All of these proposed system rewriting-based approaches can be used for resolving application ICC vulnerabilities, but using them require the necessity of modifying the code of monitored apps and the system and thereby face to legal concerns and deployment problems [42].

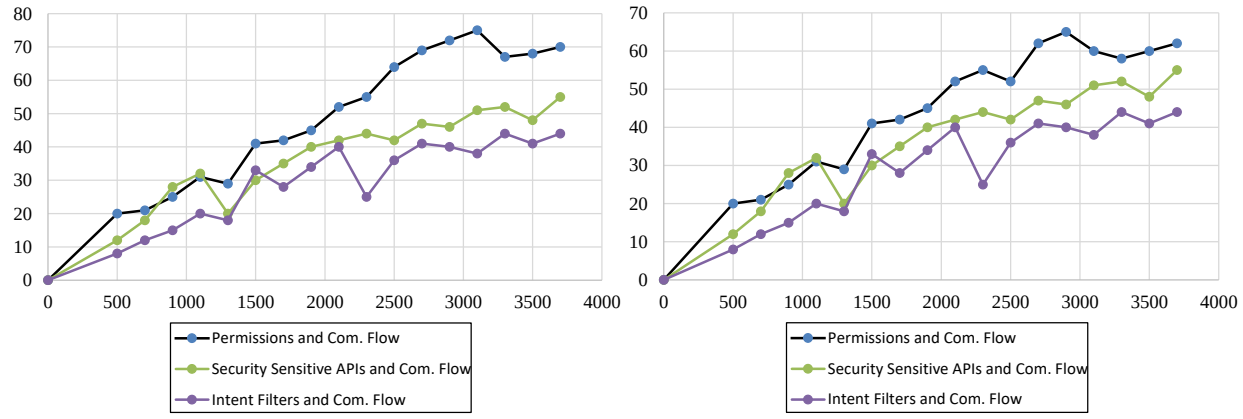
V. Conclusion and Future Work

Using machine learning methods for Android malware classification, needs using good features from applications' structure in which discriminate malicious and benign behaviors. In this paper, we proposed a new feature in Android malware classification process which in combination with other proposed features in the literature can discriminate malicious and benign applications with a good accuracy. We first showed that the flow between different applications components can be a good feature in clustering process. Then, we illustrated that we are able to achieve an accuracy as high as 85% and a false positive rate as low as 10% using SVM classifier on a data-set contain 10,320 Android malwares and benign applications using our proposed feature. However, analysing Android malwares is exposed with some challenges. A critical challenge in Android malware analysis is collecting a good data-set of Android malwares or finding the best classifier regarding to the feature attributes



(a) True Negative (b) True Positive

Fig. 3: True Positive and True Negative for different features



(a) True Negative (b) True positive

Fig. 4: True Positive and True Negative for different set of featurs combination

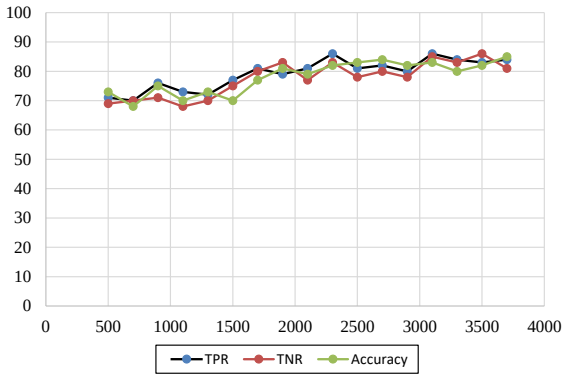


Fig. 5: Accuracy of Permission and Components flow feature set per sample size

or data set instances. As a future work, we will use different data-set and apply our classifier to them. In addition, this is very important to use different classification algorithms in training and detection phase as different algorithms may have different performance using a data-set or features.

VI. Acknowledgment

We would like to thank Morteza Amini, Rasool Jalili and anonymous reviewers for their insightful comments. We also thank Ata Chizari, Sajjad AbdollahRamezani, Mohammad Hasan Ameri and specially Mohammad Vahid Jamali.

References

[1] I. D. Corporation. (2014) Android market share reached 75% worldwide in q3 2012. [Online]. Available: http://techcrunch.com/2012/11/02/idc-androidmarket-share-reached-75-worldwidein-q3-2012 Access time: May 7, 2013

[2] M. T. Report. (2014) Third quarter 2012. [Online]. Available: http://www.mcafee.com/ca/resources/reports/rp-quarterly-threat-q3-2012.pdf Access time: May 7, 2013

[3] Y. Zhou, Z. Wang, W. Zhou, and X. Jiang, "Hey, you, get off of my market: Detecting malicious apps in official and alternative android markets," in *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, vol. 25, no. 4, 2012, pp. 50–52.

[4] M. Lindorfer, M. Neugschwandtner, L. Weichselbaum, Y. Fratantonio, V. Van Der Veen, and C. Platzer, "Andrubis–1,000,000 apps later: A view on current android malware behaviors," in *Building Analysis Datasets and Gathering Experience Returns for Security (BADGERS), 2014 Third International Workshop on.* IEEE, 2014, pp. 3–17.

[5] B. Amos, H. Turner, and J. White, "Applying machine learning classifiers to dynamic android malware detection at scale," in *Wireless communications and mobile computing conference (iwcmc), 2013 9th international*. IEEE, 2013, pp. 1666–1671.

[6] N. Peiravian and X. Zhu, "Machine learning for android malware detection using permission and api calls," in *Proceedings of the IEEE 25th International Conference on Tools with Artificial Intelligence*. IEEE, 2013, pp. 300–305.

[7] J. Sahs and L. Khan, "A machine learning approach to android malware detection," in *Proceedings of the Intelligence and Security Informatics Conference (EISIC), 2012 European*. IEEE, 2012, pp. 141–147.

[8] E. Chin, A. P. Felt, K. Greenwood, and D. Wagner, "Analyzing inter-application communication in android," in *Proceedings of the 9th international conference on Mobile systems, applications, and services*. ACM, 2011, pp. 239–252.

[9] L. Lu, Z. Li, Z. Wu, W. Lee, and G. Jiang, "Chex: statically vetting android apps for component hijacking vulnerabilities," in *Proceedings of the 2012 ACM conference on Computer and communications security*. ACM, 2012, pp. 229–240.

[10] M. I. Gordon, D. Kim, J. H. Perkins, L. Gilham, N. Nguyen, and M. C. Rinard, "Information flow analysis of android applications in droidsafe." in *Proceedings of the Network and Distributed System Security Symposium (NDSS)*. The Internet Society, 2015, pp. 6:1–6:16.

[11] F. Liu, H. Cai, G. Wang, D. D. Yao, K. O. Elish, and B. G. Ryder, "Mr-droid: A scalable and prioritized analysis of inter-app communication risks," *Proc. of MoST*, 2017.

[12] K. Xu, Y. Li, and R. H. Deng, "Iccdetector: Icc-based malware detection on android," *IEEE Transactions on Information Forensics and Security*, vol. 11, no. 6, pp. 1252–1264, 2016.

[13] L. Li, A. Bartel, T. F. Bissyandé, J. Klein, Y. Le Traon, S. Arzt, S. Rasthofer, E. Bodden, D. Octeau, and P. McDaniel, "Iccta: Detecting inter-component privacy leaks in android apps," in *Proceedings of the 37th International Conference on Software Engineering-Volume 1*. IEEE Press, 2015, pp. 280–291.

[14] C. Zheng, S. Zhu, S. Dai, G. Gu, X. Gong, X. Han, and W. Zou, "Smartdroid: an automatic system for revealing ui-based trigger conditions in android applications," in *Proceedings of the second ACM workshop on Security and privacy in smartphones and mobile devices*. ACM, 2012, pp. 93–104.

[15] A. Desnos. (2014) Androguard-reverse engineering, malware and goodware analysis of android applications. [Online]. Available: https: code.google.com/p/androguard Access time: 2013, May

[16] C. Cortes and V. Vapnik, "Support-vector networks," in *Machine learning*, vol. 20, no. 3. Springer, 1995, pp. 273–297.

[17] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg *et al.*, "Scikit-learn: Machine learning in python," in *Journal of Machine Learning Research*, vol. 12, no. Oct, 2011, pp. 2825–2830.

[18] C.-C. Chang and C.-J. Lin, "Libsvm: a library for support vector machines," in *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 2, no. 3. ACM, 2011, pp. 27:1–27:27.

[19] W. Yang, X. Xiao, B. Andow, S. Li, T. Xie, and W. Enck, "Appcontext: Differentiating malicious and benign mobile app behaviors using context," in *37th IEEE International Conference on Software Engineering*, vol. 1. IEEE, 2015, pp. 303–313.

[20] S. Pooryousef and M. Amini, "Enhancing accuracy of android malware detection using intent instrumentation," in *In Proceedings of the 3rd International Conference on Information Systems Security and Privacy (ICISSP 2017)*. Science and Technology Publications, Lda, 2017, pp. 380–388.

[21] Y. Zhang, M. Yang, B. Xu, Z. Yang, G. Gu, P. Ning, X. S. Wang, and B. Zang, "Vetting undesirable behaviors in android apps with permission use analysis," in *Proceedings of the ACM SIGSAC conference on Computer & communications security*. ACM, 2013, pp. 611–622.

[22] A. P. Felt, E. Chin, S. Hanna, D. Song, and D. Wagner, "Android permissions demystified," in *Proceedings of the 18th ACM conference on Computer and communications security*. ACM, 2011, pp. 627–638.

[23] H. Peng, C. Gates, B. Sarma, N. Li, Y. Qi, R. Potharaju, C. Nita-Rotaru, and I. Molloy, "Using probabilistic generative models for ranking risks of android apps," in *Proceedings of the 2012 ACM conference on Computer and communications security*. ACM, 2012, pp. 241–252.

[24] VirusShare. (2014) An online data set of malwares. [Online]. Available: https:virusshare.com Access time: 2015, May

[25] ISBX. (2014) An online data set of malwares. [Online]. Available: http://www.unb.ca/research/iscx/dataset/ Access time: 2015, May

[26] E. Chin, A. P. Felt, K. Greenwood, and D. Wagner, "Analyzing inter-application communication in android," in *Proceedings of the 9th international conference on Mobile systems, applications, and services*. ACM, 2011, pp. 239–252.

[27] S. J. A. B. E. B. J. K. D. Octeau, P. McDaniel and Y. L. Traon., "Effective inter-component communication mapping in android with epicc: An essential step towards holistic security analysis," in *22 nd USENIX Security Symposium*. USENIX Association, 2013, pp. 543–558.

[28] F. Wei, S. Roy, X. Ou *et al.*, "Amandroid: A precise and general inter-component data flow analysis framework for security vetting of android apps," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2014, pp. 1329–1341.

[29] L. Li, A. Bartel, T. F. Bissyandé, J. Klein, and Y. Le Traon, "Apkcombiner: Combining multiple android apps to support inter-app analysis," in *IFIP International Information Security Conference*. Springer, 2015, pp. 513–527.

[30] K. Yang, J. Zhuge, Y. Wang, L. Zhou, and H. Duan, "Intentfuzzer: detecting capability leaks of android applications," in *Proceedings of the 9th ACM symposium on Information, computer and communications security*. ACM, 2014, pp. 531–536.

[31] R. Hay, O. Tripp, and M. Pistoia, "Dynamic detection of inter-application communication vulnerabilities in android," in *Proceedings of the 2015 International Symposium on Software Testing and Analysis*. ACM, 2015, pp. 118–128.

[32] Y. Zhang, M. Yang, G. Gu, and H. Chen, "Finedroid: Enforcing permissions with system-wide application execution context," in *International Conference on Security and Privacy in Communication Systems*. Springer, 2015, pp. 3–22.

[33] C. Yang, Z. Xu, G. Gu, V. Yegneswaran, and P. Porras, "Droidminer: Automated mining and characterization of fine-grained malicious behaviors in android applications," in *European Symposium on Research in Computer Security*. Springer, 2014, pp. 163–182.

[34] Y. Aafer, W. Du, and H. Yin, "Droidapiminer: Mining api-level features for robust malware detection in android," in *International Conference on Security and Privacy in Communication Systems*. Springer, 2013, pp. 86–103.

[35] S. Chakradeo, B. Reaves, P. Traynor, and W. Enck, "Mast: Triage for market-scale mobile malware analysis," in *Proceedings of the sixth ACM conference on Security and privacy in wireless and mobile networks*. ACM, 2013, pp. 13–24.

[36] I. Burguera, U. Zurutuza, and S. Nadjm-Tehrani, "Crowdroid: behavior-based malware detection system for android," in *Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices*. ACM, 2011, pp. 15–26.

[37] G. Suarez-Tangil, J. E. Tapiador, P. Peris-Lopez, and J. Blasco, "Dendroid: A text mining approach to analyzing and classifying code structures in android malware families," in *Expert Systems with Applications*, vol. 41, no. 4. Elsevier, 2014, pp. 1104–1117.

[38] 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 *Information Security (Asia JCIS), 2012 Seventh Asia Joint Conference on*. IEEE, 2012, pp. 62–69.

[39] W. Enck, P. Gilbert, S. Han, V. Tendulkar, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth, "Taintdroid: An information-flow tracking system for realtime privacy monitoring on smartphones," in *ACM Transactions on Computer Systems (TOCS)*, vol. 32, no. 2. ACM, Jun. 2014, pp. 5:1–5:29.

[40] S. Pooryousef and S. Amini, "Fine-grained access control for hybrid mobile applications in android using restricted paths," in *Information Security and Cryptology (ISCISC), 2016 13th International Iranian Society of Cryptology Conference on*. IEEE, 2016, pp. 85–90.

[41] M. Georgiev, S. Jana, and V. Shmatikov, "Breaking and Fixing Origin-based Access Control in Hybrid Web/Mobile Application Frameworks," in *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, 2014, pp. 1–15.

[42] M. Backes, S. Bugiel, C. Hammer, O. Schranz, and P. von Styp-Rekowsky, "Boxify: Full-fledged app sandboxing for stock android." in *USENIX Security Symposium*, 2015, pp. 691–706.