Detecting Android Malware By Using A Machine Learning Ensemble Method

H. Alain Pimentel
Southern Adventist University, Collegedale, TN 37315–0370, hpimentel@southern.edu

Abstract— **Android has become the most popular mobile operating system in recent years. As its popularity has increased, so have the number of attacks to the platform. Samples of malware have been found in different popular Android apps markets, including the Google Play store. Most anti-virus software uses a signature-based approach to detect malware, however, it fails to detect unknown malware. Different techiniques have been proposed to detect unknown malware, one of which is using machine learning. This research focuses on using a combination of existing feature extraction methods and the use of an ensemble classifier to improve the accuracy of malware detection in Android. A proposal is given on how to do the feature extraction, which ensemble classifer should be used, how to train the classifier, and how to evaluate the results.**

Index Terms— **Malware Detection; Machine Learning; Android; Permissions; API calls; Smartphone Security; Classification; Ensemble Learning**

I. Introduction

By 2016, the number of smartphone users will surpass 2 billion and by 2017, 65% of the U.S. population will own a smartphone [1]. Android has become the most popular mobile OS worldwide [2]. Also, Android is available in more than 190 countries around the world and users download more than 1.5 billion apps from the Google Play Store every month [3]. The popularity of Android has brought the attention of malware apps in both official and unofficial apps markets. In 2013, a study claimed that 98.05% of all malware detected targeted Android and most were designed to steal user's money [4]. In 2014, Allix et al. [5] collected over 500,000 applications from the most popular Android apps markets and found infected apps in every apps market (with less frequency in the Google Play Store). Moreover, Maier et. al. successfully uploaded malware to the Google Play Store, passing all of Google's tests [6].

In order to identify if an Android app is malware or not, researchers have used machine learning classification algorithms to classify apps. Following this research we propose a method to improve on the accuracy shown in previous research [7], [8]. Our proposal uses an ensemble classifier to improve accuracy of malware detection in Android. Specifically, we propose the use of Troika [9], an ensemble classifier that has been proven to perform better than a single classifier. By combining Troika and using a proven feature extraction, we aim to achieve better accuracy when detecting unknown Android malware.

The rest of the paper is organized as follows. Section 2 introduces different research on detecting malware. Section 3 provides the proposed method we are going to follow to improve malware detection in Android. Section 4 concludes the paper.

II. Background

Most anti-virus software use a signature-based approach to detect malware. While they are highly effective at detecting known malware, unknown malware is not detected and systems can suffer from malware attacks. The research community has been actively looking for different ways to detect Android malware. One of the techniques used is using machine learning classification algorithms. Machine learning can be used to classify an arbitrary set into different categories. In the malware case, malicious and benign files are represented by a vector of features that can be extracted in different ways—most commonly from the binary code of an application. These feature vectors can be used to train a classifier that can be used to classify new files based on its features into benign or malware [8]. In machine learning this is considered an instance of supervised learning since a set of correctly identified files is available. This type of classification is not limited to binary cases (e.g., *malware* or *benign*), but can also be used in multi-class cases.

The accuracy of a machine learning classifier is heavily influenced by the classifier used and the type of features used [8]. The research done on malware detection is summarized in Table I.

A. Dynamic Malware Detection

Some of the constraints of collecting features dynamically on mobile devices are power consumption, memory, CPU, and limited storage. Thus, it is very important to find the best combination of extracted features and classifier to reduce the impact it has on processing complexity. Mas'ud et. al. evaluates five different sets of features using five different classifiers to find the best combination [11]. They collect data by collecting system calls from real devices. They then use these logged system calls to generate five sets including: a set with all system calls, a set with reduced features using feature reduction methods, and three sets used in previous research. Their results show that the set generated by the feature reduction methods has the best accuracy of 83%. Feature reduction can have an impact on the classifiers' performance.

Mobile sandboxes can also be used to detect malware dynamically. Sandboxes can run code and automated tools can then be used to trace what the app is doing, however

TABLE I
RESEARCH SUMMARY

Category	Advantages	Disadvantages	Research
Feature Extraction			
Dynamic	Data can be collected on real devices and at run time. Can be used to detect malware that evolves after being installed.	It is constrained by the device's capabilities. In Android, apps cannot scan other apps during run time. An app can identify if it is being run in a sandbox.	[6] [10], [11]
Static	Data can be collected off devices. The analysis is not limited by the device's capabilities.	Only analyzes code that is present on installation.	[7], [12], [13]
Machine Learning			
Binary Class	The extraction of features and classification is done off-device. Fast performance. Can detect unknown malware. 1 or 2 classes are needed for the classification. Lower computational complexity.	It can only detect up to 2 classes. Accuracy depends on the classifier and the type of features used. A diverse set of features is needed.	[7], [12], [13], [14], [15], [16], [17]
Multi Class	It can identify more than 2 classes. It provides more information on the type of malware detected.	It needs a dataset with its known classes to train the algorithm. Higher computational complexity.	[8], [18], [19]
Ensemble	Improves the performance of malware detection by combining the results of multiple classifiers. Stronger learner than single classifiers.	It requires different types of classification algorithms to minimize errors caused by a family of classifiers. Classifier must disagree in some inputs. Higher computational complexity.	[8], [18], [20]

these sandboxes have been proven to be vulnerable to certain types of malware. Maier et. al. found that *Divide-and-Conquer* attacks use a combination of dynamic code loading and fingerprinting to avoid malware scanners [6]. In order to improve sandboxes, the authors suggest using machine learning algorithms that can evolve from a couple manually analyzed samples. Other approaches that do not involve machine learning are logged behavior sequence, system calls, dynamic tainting data and control flow, and power consumption. Zhou et. al. identify certain behaviours of unknown malware using a heuristics-based filtering scheme [10]. They used Android permissions that are required by malware to cause damage. Then they are matched with specific behavioral footprints of malware. Using this approach, they were able to detect 211 malware apps from 204,040 apps collected from five apps markets. The fact that Android permissions can be used to identify malware apps, is used by several researches that approach malware detection statically.

B. Static Malware Detection

To use machine learning classification methods statically, the application file must be represented by features that are extracted from the file. These features are used by classifiers that learn any patterns that can be used to predict the class of new files [12]. One of the most common places to extract features from is the Android Manifest. Each Android API contains a certain number of permissions that an app has to declare in order to access some resources from the device. Peiravian and Zhu compared the relationship between the requested permissions by malware and benign apps. Their results showed that the average number of requested permissions by malware apps is 13.7 and for benign is 9.1 [7]. Liu and Liu also studied the effect of permissions in malware, and showed the most used permission by malware, the top three were `CALL_PHONE` `SEND_SMS` and `CHANGE_WIFI_STATE` [13]. They also listed the most common pairs of permission used by malware, where `INTERNET` and `READ_SMS` was the most popular. The use of permissions to extract features is common among other research done on Android malware detection [12], [14], [15], [16], [17].

To increase the accuracy of the classifiers, features must be diverse. For this reason, not all features should come from the manifest. Also, if the classifier uses a large number of features, it is possible that some of these features might cause more harm than good by misleading the learning algorithm, overfitting, and increasing model complexity [12]. To prevent these problems, feature reduction methods can be used to only use a certain number of top feautures [8], [12]. In 2010, Shabtai et. al. provided a method that uses permissions, feature reduction, and machine learning classifiers to classify games and tools. They claimed that this method could be used to distinguish between malware and benign apps. Recent Android malware research is similar to the method of Shabtai. Zhou and Jiang's malware dataset is also used in many of the Android malware research [7], [13], [17].

To improve perfomance, Peiravian and Zhu use a combination of permissions and API calls from the Android API 17. They also eliminate apps that contain the same feature vector to improve the accuracy of the algorithm. Their results show an accuracy of 96.39% with the combination of permissions and API calls along with the Bagging classifier [7]. Another way to use permissions to extract features, is by doing a combination of requested permissions and used permissions [13]. Liu and Liu use this approach in a two-layered method to detect Android malware. Their approach results in an accuracy of 98.6%. Android intents is a communication mechanism used between apps and they can be used by collaborating apps that together can activate malicious activity [14]. Idrees and Rajarajan propose using Android intents and permissions to classify malware [14]. Other successful approaches are the use of standard built-in and non-standard permissions [15], and a combination of permissions, intents, and API calls, which resulted in an accuracy of 0.9787 [17].

C. Ensemble Classifiers and Multi Classes

An ensemble classifier is a combination of multiple single classifiers, whose output can be combined in different ways in order to attempt to obtain a stronger learner. This methodology takes a consensus of several opinions to obtain a final decision. This approach assumes that there is a classifier diversity [18], [8], that is, the assumption that single classifiers' errors are uncorrelated. By having this classifier diversity, the number of errors can be reduced by choosing the correct combination of the single classifiers. In ensemble classifiers, the output is only useful if the base classifiers disagree about some inputs, as well as having a difference in the base classifiers' accuracy. [8]

The performance of ensemble classifiers has been studied and researched to see if they are more accurate than individual classifiers. Menahem et. al. mention that they showed a better ROC curve for the ensembles of the probabilistic neural networks (PNN) when compared to the individual best PNN [8]. Also, when evaluating ensemble classifiers using the Majority Voting schema against individual classifiers, the Majority Voting schema had better accuracy in detecting network traffic intrusions. Džeroski and Zenko mention that the stacking schema performs comparably to the best of the individual classifiers as selected by cross validation but not better [19]. However, they propose a new method for stacking that outperforms the existing stacking methods. Thus, we can see that some ensemble methods have better performance over individual classifiers.

Many approaches to classification are limited to two-class problems. However, multiple ensemble classifiers are able to classify more than two classes. Bagheri et. al. propose a new approach to ensemble classification, called generic subclass ensemble. In this method, each base classifier is trained with data belonging to a subset of classes, and the method discriminates among a subset of target categories. They give three ways to decompose original multiclass problems into smaller problems [18]. Menahem et. al. also use different ensemble to classify malware into different categories [8]. Some single classifiers can be extended to classify multiple classes, such as neural networks, decision trees, k-Nearest Neighbor, Naive Bayes, and Support Vector Machines [20].

III. Malware Detection

The process of detecting the class of an Android app starts by extracting features and creating a reduced set of features that will be used to train a classifier. Each of the stages will be described in this section (see Fig. 1 for an overview of the process). The process includes a description of the chosen classifier and how it will be trained. Certain standard metrics will be used to summarize the results of the classifier in a way that the results can be compared with previous research's results.

A. Apps Data Set

Different samples of apps are needed to train the classifier and the ratio of benign to malware apps has been chosen to match the ratio of benign to malware apps by Menahem et. al [8]. The sample apps will come from two places:

1) **Benign Apps** The Google Play Store hosts more than 1.43 million apps [21]. 3,600 apps that can be downloaded from different free categories of the Google Play Store will be selected.
2) **Malware Apps** The Android Malware Genome Project [22] has collected 1,260 malicious apps from 49 different malware families. This malware dataset has been used in most of the recent Android malware research. The latest family of malware studied in this research was found in October, 2011.

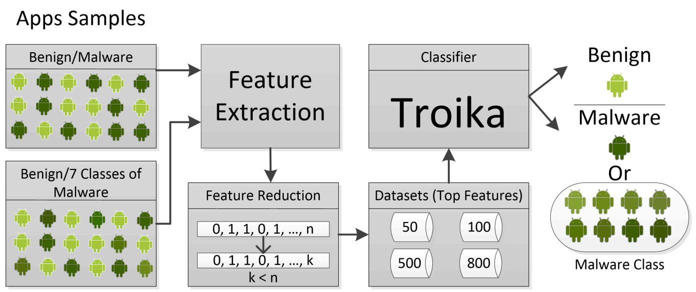

Fig. 1. Overview of the classification process [8], [7].

B. Feature Vectors

The proposed framework is a feature-based learning framework. This framework will consider the application's permissions and API calls. The feature extraction is based on the work done by Peiravian and Zhu [7] and is summarized in Fig. 2.

Peiravian and Zhu take the combination of permissions and API calls, $\{\{permissions\}, \{API\}, \{permissions{+}API\}\}$, and analyzes which combination is the best. They conclude that combining permissions and API calls improves malware detection accuracy. Following the previous conclusion, this framework will extract both the permissions and API calls into one single features vector per application.

The permissions are extracted from the *AndroidManifest.xml*. To extract the permissions, the Android APK file must be accessed. The Android APK is a compressed file containing all of the app's code (in *.dex* files), resources, assets, certificates, and the manifest itself. Once the *AndroidManifest.xml* file is extracted, the file will be parsed using the *xml.dom.minidom* package in Python. All of the extracted permissions will be converted into a binary vector, P. Now, say there are n permissions, then for any app there exists a vector $(P_1, P_2, \ldots, P_n)$, where $P_i = 0$ if and only if the i_{th} permission was not found in the *AndroidManifest.xml*. Similarly, $P_i = 1$ if and only if the i_{th} permission was found in the *AndroidManifest.xml*.

The *.dex* file, which contains the compiled Android application code, will also be extracted and decompiled to find which API calls are used by the app.. *APKtools* [23] will be used to decompile the *.dex* file. The API call will be used to generate another feature binary vector, A, following the same technique as the permissions vector. Now, say there are n API calls, then for any app there exists a vector $(A_1, A_2, \ldots, A_n)$, where $A_i = 0$ if and only if the i_{th} API call was not used in

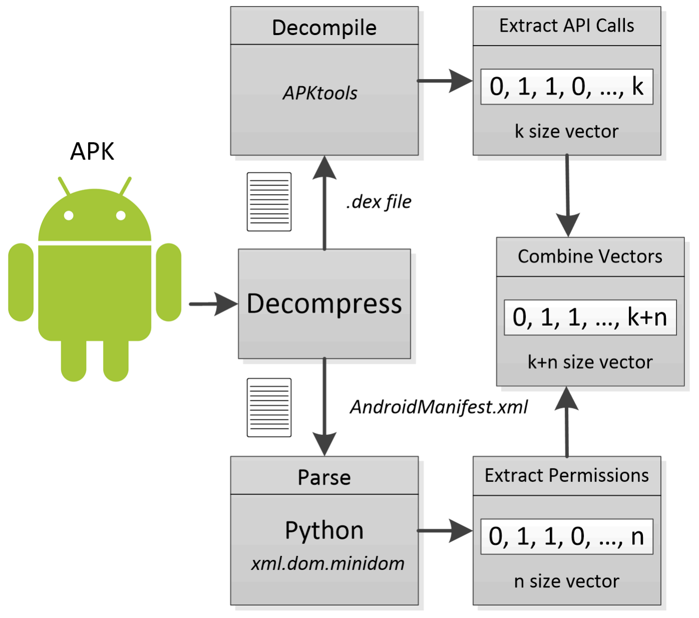

Fig. 2. Feature extraction process [7].

the app. Similarly, $A_i = 1$ if and only if the i_{th} API call was used in the app.

C. Datasets

Each app, represented by a feature vector, will be added to different datasets that can be differentiated by the number of classes and features they contain. For the first type of dataset, named *com-2C*, both vectors will be appended and a label will be added to distinguish between *Benign* or *Malware*. For the second type of dataset, named *com-8C*, both vectors will be appended, but each app can belong into 7 different subclasses of malware: *Trojan-SMS, Backdoor, Trojan, Adware, RiskTool, Trojan-Downloader, Trojan-Spy* [4]. If the app is benign, the label *Benign* will be added. Following the datasets created Peiravin and Zhu [7], we will be using the Android API 17. This will give us 130 permission features and 1,326 API features.

Since many apps belong to the same malware family, many malware will have the same feature values. To reduce repetition, any app with the same feature vector will be eliminated to leave only one sample per family [7]. This reduced the malware sample in [7] about half the original size. Now, a large number of features can become a problem since some features might not contribute anything, but instead damage the accuracy of the classification [8]. A large number of extracted features can lead to classification problems, such as misleading the learning algorithm, overfitting, reducing generality, and increasing model run-time [11]. Thus, the number of features will be reduced by applying three feature selection methods: Information Gain, Fisher Score, and Chi-Square [7], [12]. These feature-selection methods work as objective functions that evaluate features by their information content and estimate their expected contribution to the classification task, returning a score for each metric individually [12]. The top 50, 100, 500, 800 features that had the highest rank are selected to avoid bias while selecting top features [12], [7]. Thus, both *com-2C* and *com-8C* will have four different versions (top 50, top 100, top 500, top 800.)

The datasets are summarized in Table II.

TABLE II
PROPERTIES OF THE DATASETS PROPOSED

Dataset	No. Classes	No. Instances	No. Features
com-T50-2C	2	< 4860	50
com-T100-2C	2	< 4860	100
com-T500-2C	2	< 4860	500
com-T800-2C	2	< 4860	800
com-T50-8C	8	< 4860	50
com-T100-8C	8	< 4860	100
com-T500-8C	8	< 4860	500
com-T800-8C	8	< 4860	800

D. Classifier

Troika [9] was designed to address certain problems with stacking, such as its poor performance with multi-class problems. Troika addresses the multi-class problem by using three layers of combining classifiers, rather than one.

In [8] Menahem et. al. use different combining methods to detect malware in terms of accuracy, AUC, and execution time. They compared the following combination methods: best classifier, majority voting, performance weighting, distribution summation, Bayesian combination, Naïve Bayes, stacking, and Troika. They use five different base classifiers: C4.5, KNN, VFI, OneR, and Naïve-Bayes. As mentioned before, they take into consideration classifier diversity, and so each base classifier belongs to a different family. They experimented on an unbalanced dataset of 22,735 benign files and 7,960 malware that were identified as Win32 executables and DLLs.

Having a large number of features can harm the accuracy of the classification since some of those features might not contribute anything. Thus, they chose to shorten the number of features used by extracting the top features using the Document Frequency. Then they used three methods of feature selection including Document Frequency, Fisher Score, and Gain Ratio. They continued to test on 5 different datasets, including versions of binary classes (benign, malicious) and 8 classes (Virus, Worm, Flooder, Trojan, Backdoor, Constructor, Nuker, benign.)

To evaluate the performance of the individual classifiers and ensemble classification methods, they used Accuracy, Area Under the ROC curve, and Training Time. Their results show Troika was the best in terms of accuracy and AUC. In both AUC and Accuracy, Troika is better than the best-base classifier, showing again that using Troika would improve over current Android malware detection research. It is also better than stacking, the method Troika is trying to improve over. Also, Troika had an improvement of 8% on the AUC when using a multi-class dataset, which makes it a good option to detect different classes of malware.

For execution time, the meta-combiner ensembles (stacking and Troika) performed the worst. This was to be expected since meta-combiners take more time during the training phase. Since the purpose of this research is to improve the accuracy of malware detection in Android and detecting multi-class malware, Troika will serve as the classifier.

E. Training

For Troika, the base classifiers and the meta-data set are trained. Since Troika is based on stacking, training sets have more partitions due to their inner k-fold cross-validation training process. Troika uses 72% of the data set as the train-set. Of the original train-set, Troika uses 80% of it to train the base classifiers. The other 20% is used to train the combining classifiers. To have a general depiction of accuracy, the evaluation will be performed in 10-fold cross-validation format repeated 5 times [12], [8]. To follow the experiment of Menahem et. al. [8] closely, the training set will be randomly partitioned into 10 disjoint instance subsets, for each cross-validation iteration. Where each subset will be used once as a test set and nine times as a training set.

F. Standard Metrics

True Positive Rate (TPR), *False Positive Rate* (FPR), Accuracy, Area Under the ROC curve, and Training Time will be used in order to have a basis to compare this experiment with previous Android Malware research.

- **TPR** is the rate of correctly detecting a feature vector as malware.

  $$TPR = \frac{TP}{TP + FN}$$

  TP (True Positives) is the number of malware instances correctly classified.
- **FPR** is the rate of false detection of a benign app as malware. FN (False Negatives) is the number of malware instances incorrectly classified.

  $$FPR = \frac{FP}{FP + TN}$$

  FP (False Positives) is the number of benign instances incorrectly classified as malware. TN (True Negative) is the number of benign instances correctly classified.
- **Accuracy** is the rate of correctly classified malware and benign instances over the total population.

  $$ACC = \frac{TP + TN}{TP + FN + FP + TN}$$

- **Area Under the ROC Curve** The ROC curve is a graph of the relation of *TPR* vs *FPR*. An *Area Under the Curve* closer to 1 indicates that the classifier will obtain 0% false positives and 100% true positives.
- **Training Time** To quantify the CPU usage of the algorithm and its energy usage.

To have a good performance, the classifier must have a high *TPR* and a low *FPR*.

IV. Conclusions

In this paper we provided a summary of state of the art research of Android malware detection. Also, we proposed a method to detect known and unknown Android malware by using a machine learning ensemble method. To build the models, we extract features from the Android *.apk* by getting the permissions and API calls. As a classifier we use Troika, an ensemble method that uses several classifiers to improve performance over a single classifier. We also propose doing two classifications: *malware vs. benign* and *classes of malware vs benign*. Our method includes guidelines on how to do the feature extraction and test the algorithms using several datasets. It also includes guidelines on how to judge the performance of the method.

For future work, we will implement this method and compare it to the existing research.

References

[1] eMarketer. (2014) 2 billion consumers worldwide to get smart(phones) by 2016. [Online]. Available: http://www.emarketer.com/Article/2-Billion-Consumers-Worldwide-Smartphones-by-2016/1011694

[2] J. Hahn. (2015) Android claims 81.52014, ios dips to 14.8http://www.digitaltrends.com/mobile/worldwide-domination-android-and-ios-claim-96-of-the-smartphone-os-market-in-2014/

[3] Android. (2015) Android, the world's most popular mobile platform. [Online]. Available: http://developer.android.com/about/index.html

[4] R. U. Victor Chebyshev. (2014) Mobile malware evolution: 2013. [Online]. Available: http://securelist.com/analysis/kaspersky-security-bulletin/58335/mobile-malware-evolution-2013/

[5] K. Allix, Q. Jerome, T. Bissyande, J. Klein, R. State, and Y. le Traon, "A forensic analysis of android malware – how is malware written and how it could be detected?" in *Computer Software and Applications Conference (COMPSAC), 2014 IEEE 38th Annual*, July 2014, pp. 384–393.

[6] D. Maier, T. Muller, and M. Protsenko, "Divide-and-conquer: Why android malware cannot be stopped," in *Availability, Reliability and Security (ARES), 2014 Ninth International Conference on*, Sept 2014, pp. 30–39.

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

[8] E. Menahem, A. Shabtai, L. Rokach, and Y. Elovici, "Improving malware detection by applying multi-inducer ensemble," *Computational Statistics & Data Analysis*, vol. 53, pp. 1483–1494, 2009.

[9] E. Menahem, L. Rokach, and Y. Elovici, "Troika–an improved stacking schema for classification tasks," *Information Sciences*, vol. 179, no. 24, pp. 4097–4122, 2009.

[10] W. Z. Yajin Zhou, Zhi Wang and X. Jiang, "Hey, you, get off of my market: Detecting malicious apps in official and alternative android markets," in *Proceedings of the 19th Annual Network and Distributed System Security Symposium, NDSS*, Feb 2012.

[11] M. Mas'ud, S. Sahib, M. Abdollah, S. Selamat, and R. Yusof, "Analysis of features selection and machine learning classifier in android malware detection," in *Information Science and Applications (ICISA), 2014 International Conference on*, May 2014, pp. 1–5.

[12] A. Shabtai, Y. Fledel, and Y. Elovici, "Automated static code analysis for classifying android applications using machine learning," in *Computational Intelligence and Security (CIS), 2010 International Conference on*, Dec 2010, pp. 329–333.

[13] X. Liu and J. Liu, "A two-layered permission-based android malware detection scheme," in *Mobile Cloud Computing, Services, and Engineering (MobileCloud), 2014 2nd IEEE International Conference on*, April 2014, pp. 142–148.

[14] F. Idrees and M. Rajarajan, "Investigating the android intents and permissions for malware detection," in *Wireless and Mobile Computing, Networking and Communications (WiMob), 2014 IEEE 10th International Conference on*, Oct 2014, pp. 354–358.

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

[16] S. Yerima, S. Sezer, G. McWilliams, and I. Muttik, "A new android malware detection approach using bayesian classification," in *Advanced Information Networking and Applications (AINA), 2013 IEEE 27th International Conference on*, March 2013, pp. 121–128.

[17] 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*, Aug 2012, pp. 62–69.

[18] M. Bagheri, Q. Gao, and S. Escalera, "Generic subclass ensemble: A novel approach to ensemble classification," in *Pattern Recognition (ICPR), 2014 22nd International Conference*, Aug 2014, pp. 1254–1259.

[19] S. Džeroski and B. Ženko, "Is combining classifiers with stacking better than selecting the best one?" *Mach. Learn.*, vol. 54, no. 3, pp. 255–273, Mar 2004.

[20] M. Aly, "Survey on multiclass classification methods," in *CalTech*, Nov 2005.

[21] G. Sterling. (2015) Report: Google play finally passes ios app store in number of apps, developers. [Online]. Available: http://marketingland.com/report-google-play-finally-passes-ios-app-store-number-apps-developers-114115

[22] Y. Zhou and X. Jiang, "Dissecting android malware: Characterization and evolution," in *Security and Privacy (SP), 2012 IEEE Symposium on*, May 2012, pp. 95–109.

[23] ApkTool. (2014) android-apktool: A tool for reverse engineering android apk files. [Online]. Available: https://code.google.com/p/android-apktool/