Android Malware Detection Using Category-Based Machine Learning Classifiers

Huda Ali Alatwi
Rochester Institute of Technology
Rochester, NY
haa4070@rit.edu

Tae Oh
Rochester Institute of Technology
Rochester, NY
thoics@rit.edu

Ernest Fokoue
Rochester Institute of Technology
Rochester, NY
epfeqa@rit.edu

Bill Stackpole
Rochester Institute of Technology
Rochester, NY
bill.stackpole@rit.edu

ABSTRACT

Android malware growth has been increasing dramatically as well as the diversity and complicity of their developing techniques. Machine learning techniques have been applied to detect malware by modeling patterns of static features and dynamic behaviors of malware. The accuracy rates of the machine learning classifiers differ depending on the quality of the features. We increase the quality of the features by relating between the apps' features and the features that are required to deliver its category's functionality. To measure the benign app references, the features of the top rated apps in a specific category are utilized to train a malware detection classifier for that given category. Android apps stores such as Google Play organize apps into different categories. Each category has its distinct functionalities which means the apps under a specific category are similar in their static and dynamic features. In other words, benign apps under a certain category tend to share a common set of features. On the contrary, malicious apps tend to have abnormal features, which are uncommon for the category that they belong to. This paper proposes category-based machine learning classifiers to enhance the performance of classification models at detecting malicious apps under a certain category. The intensive machine learning experiments proved that category-based classifiers report a remarkable higher average performance compared to non-category based.

CCS Concepts

•Security and privacy → Malware and its mitigation; Software security engineering;

SIGITE'16, September 28-October 01, 2016, Boston, MA, USA
© 2016 ACM. ISBN 978-1-4503-4452-4/16/09...$15.00
DOI: http://dx.doi.org/10.1145/2978192.2978218

Keywords

Android malawre detection; static analysis; machine learning

1. INTRODUCTION

According to International Data Corporation (IDC), Android operating system (OS) is the most popular smartphone platform with 82.2% of the market share, while 13.9% for iOS apple in the second quarter of 2015 [1]. Statistically Android is the first targeted platform by malware authors seeking to take the control of over one billion of mobile devices in the world [2]. According to F-Secure, a computer security company, Android had 97% of smartphone malware in 2014 [3].

Android is an open source development environment enables developers to deploy their own apps and distribute them through Android apps centers. Android's popularity is a result of being an open source, third-party distribution centers, a rich SDK, and Java's popularity. Because of the open environment, malaware authors can develop malicious apps that abuse the platform features or pack a legitimate app with a piece of malicious code; besides, exploiting vulnerabilities in the platform, hardware, or other installed apps to lunch malicious behaviors. Mainly, malware authors seek to access confidential user's data, get monetary benefits via premium SMS, or join the device to a botnet. Even legitimate apps introduce the risk of privacy-invading; Mcafee reported in Q1 2014 that 82% of Android apps track user's and 80% gather location data [4].

Android malware detection has three main approaches, which are static, dynamic and hybrid. Static analysis technique examines the source code of the app without execution to detect malicious codes and patterns. To accomplish this, the executable app is disassembled to the source code files from where many features are extracted such as: permissions, hardware components, broadcast receivers, APIs, intents, data flow, control flow, etc. On the other hand, dynamic analysis examines the app during run-time in a controlled virtual environment and monitors the app's dynamic behavior and the system's responses. The monitored dynamic features are network connections, system calls, resources' usage, etc. For both approaches, the data is collected to train machine learning classifiers to build a separation modeling between benign and malicious characteristics of the apps.

Normally, static analysis reports a high accuracy rate in detecting malware, and it is relatively cheap compared to dynamic analysis in terms of effort, time, and computational resources. Machine learning techniques are the current methods for detecting malware on Android. Mostly, the researches focus on training supervised machine learning classifiers to detect, classify the malware to a known malware family, or using semi-supervised learning to discover a new one. Machine learning techniques can provide remarkable detection accuracy rates depending on the quality of the features which are used for training the classifiers e.g how specific they are. Whereas the accuracy rates of the classifiers increase with increasing the quality of the features, we relate between the apps' features and the features that are needed to deliver its category's functionality to detect malicious patterns. In other words, a malware detection classifier is trained for each category separately.

In this paper, our approach utilizes the features of benign apps under a particular category to detect malware in the same category. We relate between the features that the app requests and a common set of features for its category. Usually, Android app stores organize apps into different categories. For example, Google Play organizes apps in 26 categories such as: "Health & Fitness", "News & Magazine", "Books & References", "Music & Audio", etc. Each category has its distinct functionalities, which means the apps under a certain category share a similar combination of features. One group of these features is the permissions. Permissions are the privileges that enable an app to access the system's resources to perform its functions. Each built-in permission is responsible for providing the capabilities to execute a particular process. Apps belong to a specific category deliver the same functionality and as a result, they require a common combination of permissions. For instance, apps under "Communication" category commonly request `READ_CONTACTS`, but it is uncommon if this permission is requested by apps under "Personalization". In general, benign apps under a certain category tend to have a common set of features: permissions, intents filters, hardware components, broadcast receivers, APIs, etc. On the contrary, malicious apps tend to request abnormal features, less or more than what is common for the category that they belong to. This study proposes category-based machine learning classifiers to enhance the performance of classification models at detecting malicious apps under a certain category.

2. RELATED WORK

[Arp et al. 2014] proposed (DERBIN) a lightweight static analysis framework that extracts a set of features from the app's AndroidManifest.xml (hardware components, requested permissions, App components, and filtered intents) and disassembled code (restricted API calls, used permissions, restricted API calls, network addresses) to generate a joint vector space [5]. Support Vector Machines (SVM) was applied on the dataset to learn a separation between benign and malicious apps.

[Chan and Song 2014] selected permissions and APIs features with Information Gain to train different classifiers: Naive Bayes, SVM with SMO algorithm, RBF Network, Multi Layer Perceptron, Liblinear, J48 decision tree and Random Forests [6].

[Idrees and Rajarajan 2014] trained Naive Bayes classifier with permissions and intents along with using statistic techniques Kstar and Prism to detect malicious apps [7].

[Yerima et al. 2013] used mutual information (entropy) to rank the extracted features: permissions, APIs, Linux Native commands for training a Bayesian classifier [8].

[Wu et al. 2012] proposed (Droidmat) detects malware through analyzing AndroidManifest.xml and tracing systems calls [9]. Droidmat trains a K-means classifier with the apps' permissions, components, intent messages, and API calls.

[Sanz et al 2012] trained machine learning classifiers with features extracted from the AndroidManifest.xml file, the source code files (the frequency of occurrence of the printable strings), and Android market (permissions, rating, and number of ratings) [10].The machine learning algorithms were applied: Decision Trees (DT), K-Nearest Neighbour (KNN), Bayesian Networks (BN), Random Forest (RF) and Support Vector Machines (SVM).

[Sahs and Khan 2012] trained one-class Support Vector Machines (SVM) classifier with the extracted permissions and the control flow graphs from benign apps [11].

3. DESIGN AND METHODOLOGY

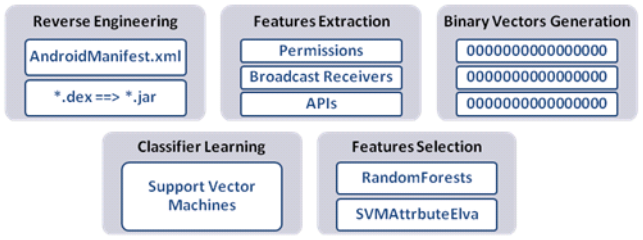

Figure 1: Category-Based Machine Learning Technique

The framework of this study, as shown in Figure 1, consists of five components. The first component is a module to reverse engineer the apps' apk files into source code in forms of AndroidManiFest.xml and java classes. The second component is a module to parse three groups of features: permissions, broadcast receivers, and APIs. The third module is to transform the extracted features into a binary vector. Each app is represented as a single instance with binary vector of features and a class label indicates whether the app is benign or malicious. The fourth component is selecting the best combination of features that can be used as predictor to increase generalization and decrease over-fitting in the learned model. The last component is modeling a SVM classifier for each of the three datasets separately.

3.1 Data Collection

Category	Benign	Malware	Total
All Categories	1000	4063	5063
Music & Audio	854	1136	1990
Personalization	732	942	1674

Table 1: Number of Apps in each dataset

The first part of the study is building a SVM classifier for apps from all categories. The classifier (all-SVM) was trained with features of malicious apps and the top rated apps from all the 26 categories on Google Play Store. The second part of our study mainly focused on relating between the apps' features and a common set of features for the category that the apps belong to. Two categories of apps have been chosen: ”Music and Audio” and ”Personalization”. The features were collected from each dataset separately to train two SVM malware detection classifiers, one (music-SVM) for ”Music and Audio” apps and the another (persona-SVM) for ”Personalization” apps. The goal is to compare between the performance of the category-based (music-SVM & persona-SVM) and non-category based (all-SVM) classifiers at detecting malicious apps in the two categories. Table 1 shows the number of the benign and malicious apps in the three datasets: all categories, ”Music & Audio”, and ”Personalization”.

3.2 Reverse Engineering

This module integrates multiple reverse engineering tools to transform the apps' .apk files into their source codes. Table 2 shows a list of tools used in the reverse engineering process.

Tool	**Use**
APKTool	Decodes .apk files into the original forms.
dex2jar	Converts .dex files into .jar files.
jd	Converts .jar files into .java files.

Table 2: Reverse Engineering Tools

3.3 Features Extraction

3.3.1 Permissions

Android runs apps in sandboxes on the virtual environment (Davik VM), where apps are isolated from the resources of the system and other apps. Android regulates apps access to the resources of hardware, OS, and other installed apps through the permissions settings. The apps need to have the appropriate permissions settings to perform privileged processes on the system. In this paper, the permissions requested by an app are compared with a common set of permissions are needed for the functionalities of the category that the app belongs to. The functionalities of a specific category require a common set of permissions. If the app requests uncommon or overprivileged permissions compared to the common permissions that requested by the benign apps in the same category that strongly indicates a malicious intention.

3.3.2 Broadcast Receivers

Android allows apps to interact with the system and other apps by sending and listening to broadcast messages. Android announces the systems events in broadcast messages that can be received by other apps when listening to specific kind of events such as (`BOOT_COMPLETED`, `SMS_RECEIVED`, `CONNECTIVITY_CHANGE`, etc...). Apps also can send broadcast messages to other apps to trigger some activities such as opening and activating a web page. The broadcast receivers that listen to the Android system events are the only ones used as features in training the classifiers. In this paper, the app's broadcast receivers are compared with the Android broadcast events that the top rated apps in the same category they listen to.

3.3.3 APIs

APIs are classes and interferes that enable apps to interact and lunch the functionalities of the underlying Android system. Android platform provides a hierarchical structure of classes based on the targeted version of the Android system which specified by the API level. The APIs are used as features to identify the processes that the app wants to execute. The malicious apps call sensitive APIs that enable lunching malicious activities such as loading external jar files by calling `loadClass()` or collecting device's info by calling `getDeviceId()`. Apps under a certain category call a certain set of APIs that are needed for providing the category's functionalities. In this paper, the APIs requested by the app's are compared with a common set of APIs that requested by the benign apps in the same category.

3.4 Binary Vector Generation

Each app in the sample was represented as a single instance with a binary vector of features and a class label indicates whether the app is benign or malicious. If the feature is present in the app it is represented by 1, if it is not present in the app, it is represented by 0.

3.5 Features Selection

This step aims to reduce the high-dimensional of the variables space in the datasets by identifying subsets of features that are the best predictors for the class labels of the datasets' instances. Generally, features selection enhances the generalization of the learned models by reducing overfitting. It also increases the classification accuracy, reduces the training and classification times, and produces simplified interpreted models. The Randomforests algorithm was used to select a subset of features for each dataset. Table 3 shows the number of the selected features for each dataset.

Features	**Permission**	**Recevier**	**API**
All Categories	142	136	2497
Music	57	84	2284
Personalization	56	92	1776

Table 3: Number of Selected Features

3.6 Support Vector Machines Classifier

Support Vector Machines (SVM) is a representation of the training data as points in the space that conglomerate based on their category in form of groups that are separated by a clear distinct gap called a hyperplane. In the training phase, SVM builds up a model of patterns from the training data which is used as a space for classification phase. In the classification phase, the new input points are mapped into the trained space and categorized based on which side of the gap they fall on. SVM forms a hyperplane or set of hyperplanes for data classification and regression. For more confidence and less generalization error, the hyperplane must be selected by a functional margin that makes the distance between the nearest training data points in any class as much larger as possible. The SVM was selected because its resistance to over-fitting even in the very high dimensional variables space like our datasets.

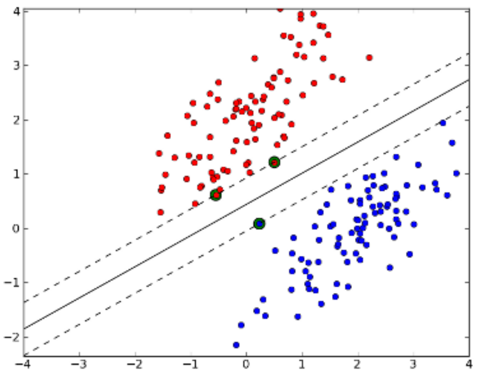

Figure 2: Support Vector Machines

4. EXPERIMENTS AND RESULTS

4.1 Experiments' Settings

For each of the three datasets: allCateg, musicCateg and personaCateg, 70% of the dataset was used for training the classifier, and 30% for testing. The datasets were randomly shuffled in each round of the 50 iterations that are used to average the performance of the classifiers.

4.2 Evaluation Measurements

In order to evaluate the performance of the classification models, the following metrics were used.

- **Accuracy:** The proportion of the total number of the apps that are correctly classified whether as benign or malicious.
$$\text{Accuracy} = \frac{tp + tn}{tp + tn + fp + fn}$$
- **Precision:** The proportion of the actual malicious apps are correctly classified to the total of all apps that are classified as malicious.
$$\text{Precision} = \frac{tp}{tp + fp}$$
- **Recall:** The proportion of the malicious apps that are classified correctly to the total number of the malicious that are classified correctly as malicious or incorrectly as benign.
$$\text{Recall} = \frac{tp}{tp + fn}$$
- **F-Measure:** The harmonic mean of precision and recall. This value tells how much the model is discriminative.
$$\text{F-Measure} = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$
- **True Positive (TP):** The number of the malicious apps that are correctly classified as malicious.
- **False Negative (FN):** The number of the malicious apps that are incorrectly classified as not malicious (benign)
- **True Negative (TN):** The number of the benign apps that are correctly classified as not malicious (benign).
- **False Positive (FP)**: The number of the benign apps that are incorrectly classified as malicious.

4.3 Testing "Music & Audio" Apps with musicCateg & allCateg Classifiers

Metric	musicCateg-SVM	allCateg-SVM
Accuracy	0.9872	0.9458
Precision	0.9777	1
F-Measure	0.9886	0.9547
Recall	0.9999	0.9134
FPR	0.0286	0
TPR	0.9999	0.9134
FNR	0.0002	0.0865
TNR	0.9713	1
Specificity	0.9713	1
Sensitivity	0.9999	0.9134

Table 4: Testing music apps with musicCateg & allCateg SVM classifers

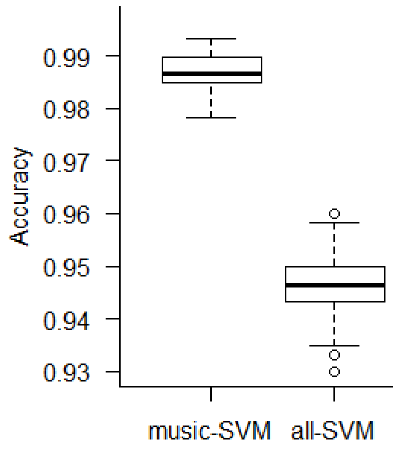

Figure 3: Accuracy of "musicCateg" &"allCateg" Classifiers

Figure 3 shows the variation of the category-based (musicCateg-SVM) and non-category based (allCateg-SVM) classifiers' accuracy at detecting malicious music apps over 50 iterations. As it indicated in Table 4, the musicCateg-SVM classifier reports average accuracy 0.9872 higher than allCateg-SVM classifier which reports 0.9458. Figure 4 shows the variation of the classifiers' Fmeasure over 50 iterations, as well. As it indicated in Table 4, the musicCateg classifier reports average F-measure 0.9886 higher than allCateg classifier which reports 0.9547. The boxplots of accuracy and F-measure of the classifiers in Figure 3 and Figure 4 show noticeable differences in the average performance of category-based (musicCateg) and non-category based (allCateg) classifiers at detecting malicious in the "Music & Audio" category. Likewise, the T.test function produces 4.18% a mean of the differences which proves that musicCateg-SVM outperforms allCateg-SVM. We used paired t-test function to determine whether the mean of the Fmeasure of the music-SVM classifier and all-SVM classifer are equal to each other. The null hypothesis is that the two means are equal, and the alternative is that they are not.

In this case, the null hypothesis and the alternative as following:

$$H_0 : \mu^{Fmeasure}_{musicCateg} \leq \mu^{Fmeasure}_{allCateg}$$

$$H_a : \mu^{Fmeasure}_{musicCateg} > \mu^{Fmeasure}_{allCateg}$$

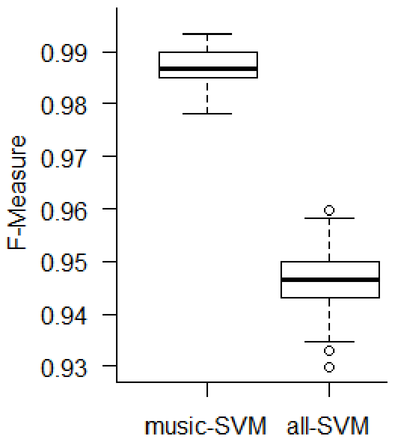

Figure 4: Fmeasure of musicCateg & allCateg Classifiers

Over m=50 replications of the calculation of the F-measure, an upper sided paired t-test at the alpha=0.05 significance level yielded an essentially zero Pvalue and an observed difference of 3.4% between the average Fmeasure for the design based on music features only and the design based on all the features. The 95% lower confidence bound on the difference of the two average Fmeasures was also found to be 3.5%, so that we are 95% confident that when using the SVM classifier on our data, the average F-measure for the design based on musical features only, exceeds the average F-measure based on all the features by a minimum of 3.5%. 50 replication of calculations of F-measures under both conditions on the same replicate of dataset. Hence: the use of a Paired t.test and from the boxplots in Figure 4, our assumption gaussianity is clearly plausible.

Empirically:

$$\mu^{Fmeasure}_{musicCateg} - \mu^{Fmeasure}_{allCateg} = 3.4\%$$

Informally,

$$\mu^{Fmeasure}_{musicCateg} > \mu^{Fmeasure}_{allCateg}$$

Theoretical by inference:$p - value \simeq 0$ rejects H_0
Concludes:

$$H_a : \mu^{Fmeasure}_{musicCateg} > \mu^{Fmeasure}_{allCateg}$$

$$\mu^{Fmeasure}_{musicCateg} - \mu^{Fmeasure}_{allCateg} \epsilon \, [3.25, \ \infty]$$

$$LB_{95\%}(\mu^{Fmeasure}_{musicCateg} - \mu^{Fmeasure}_{allCateg}) = 3.25\%$$

4.4 Testing "Personalization" Apps with personaCateg & allCateg SVM classifers

Metric	personaCateg-SVM	allCateg-SVM
Accuracy	0.9855	0.8947
Precision	0.9833	1
F-Measure	0.9736	0.9318
Recall	0.9651	0.8726
FPR	0.0006	0
TPR	0.9651	0.8726
FNR	0.0348	0.1273
TNR	0.9934	1
Specificity	0.9934	1
Sensitivity	0.9651	0.8726

Table 5: Testing Personalization apps with personaCateg & allCateg SVM classifiers

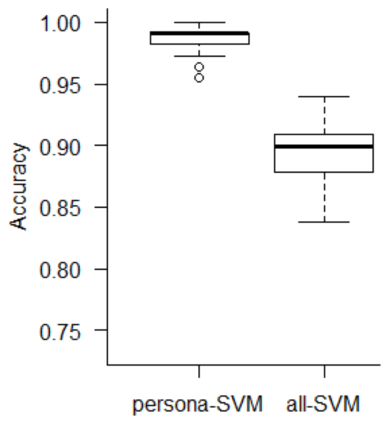

Figure 5: Accuracy of "personaCateg" &"allCateg" Classifiers

Figure 5 shows the variation of the category-based (personaCateg-SVM) and non-category based (allCateg-SVM) classifiers' accuracy at detecting malicious personalization apps over 50 iterations. As it indicated in Table 5, the personaCateg-SVM classifier reports average accuracy 0.9855 higher than allCateg-SVM classifier which reports 0.8947. Figure 6 shows the variation of the classifiers' F-measure over 50 iterations, as well. As it indicated in Table 5, the personaCateg classifiers reports average F-measure 0.9736 higher than allCateg classifier 0.9318. The boxplots of accuracy and f-measure of the classifiers in Figure 5 and Figure 6 show noticeable differences in the average performance of category-based (personaCateg) and non-Category based (allCateg) classifiers at detecting malicious apps in the "Personalization" category. Likewise, the T.test function produces 4.18% a mean of the differences which proves that personaCateg-SVM outperforms allCateg-SVM.

5. CONCLUSION

The performance of category-based classifiers were compared with non-category based classifiers at detecting malicious apps under a specific category. Three datasets of apps' features were collected from malware and top rated

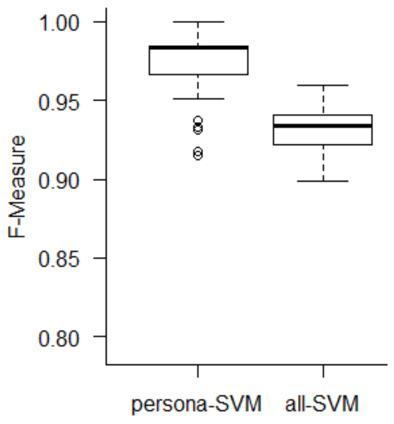

Figure 6: Fmeasure of ”personaCateg” &”allCateg” Classifiers

apps on the Google Play Store: all categories apps, ”Music & Audio” apps, and ”Personalization” apps. For each dataset, a SVM classifier with three groups of features: permissions, broadcast receivers, and APIs. For testing, the apps from ”Music & Audio” were tested with (musicCateg-SVM) and (allCateg-SVM) classifiers, respectively, and the apps from ”Personalization” were tested with (personaCateg-SVM) and (allCateg-SVM) classifiers, as well. The category-based classifiers reported a higher performance compared to the non-category based at detecting malicious and benign in the two categories. Whereas the accuracy rates of the machine learning classifiers increase with increasing the quality of the features, we related between the apps' features and the features that are required to deliver its category's functionality. The features of the top rated apps in a specific category were utilized to train a malware detection classifier for that given category. We improved the performance of the machine learning classifiers by increasing the quality of the features and considering the app category in building the classifiers.

For the future work, we will consider three aspects. First, we will consider adding other static features such as: functions calls in training the classifiers to have a better understanding of the processes that apps may activate to increase the detection accuracy of the classifiers. Second, the proposed solution could be implemented in a large-scale by building profile models for other categories and sub categories. Third, our solution could be integrated with dynamic detection techniques by profiling dynamic features for each category.

6. REFERENCES

[1] Smartphone OS Market Share, 2015 Q2. (August 2015). Retrieved December 19, 2015 from http://www.idc.com/prodserv/smartphone-os-market-share.jsp.

[2] Alistair Barr. 2015. Google Says Android Has 1.4 Billion Active Users. (September 2015). Retrieved December 19, 2015 from http://www.wsj.com/articles/google-says-android-has-1-4-billion-active-users-1443546856.

[3] Gordon Kelly . 2014. Report: 97% of mobile malware is on android. (March 2014). Retrieved December 19, 2015 from http://www.forbes.com.

[4] McAfee Consumer Mobile Security Report. (February 2014). Retrieved December 19, 2015 from http://www.mcafee.com/us/resources/reports/rp-mobile-security-consumer-trends.pdf.

[5] Arp, D., Spreitzenbarth, M., Hübner, M., Gascon, H., Rieck, K., and Siemens, C. (2014). Drebin: Effective and explainable detection of android malware in your pocket. In *Proc. of NDSS*.

[6] Wu, D.-J., Mao, C.-H., Wei, T.-E., Lee, H.-M., and Wu, K.-P. (2012). Droidmat: Android malware detection through manifest and api calls tracing. In *Information Security (Asia JCIS), 2012 Seventh Asia Joint Conference on*, pages 62–69. IEEE.

[7] Sanz, B., Santos, I., Laorden, C., Ugarte-Pedrero, X., and Bringas, P. G. (2012). On the automatic categorisation of android applications. In *Consumer Communications and Networking Conference (CCNC), 2012 IEEE*, pages 149–153. IEEE.

[8] Sahs, J. and Khan, L. (2012). A machine learning approach to android malware detection. In *Intelligence and Security Informatics Conference (EISIC), 2012 European*, pages 141–147. IEEE.

[9] Chan, P. P. K. and Song, W. (2014). Static detection of Android malware by using permissions and API calls. In *International Conference on Machine Learning and Cybernetics, 2014 Lanzhou*, pages 82–87. IEEE.

[10] Idrees, F. and Rajarajan, M. (2014). Investigating the android intents and permissions for malware detection. In *10th International Conference on Wireless and Mobile Computing, Networking and Communications, 2014 Larnaca*, pages 354–358. IEEE.

[11] S. Y. Yerima and S. Sezer and G. McWilliams and I. Muttik (2013). A New Android Malware Detection Approach Using Bayesian Classification. In *Advanced Information Networking and Applications (AINA), 2013 IEEE 27th International Conference on, 2013 Barcelona*, pages 121–128. IEEE.