ADROIT: Android malware detection using meta-information

Alejandro Martín*, Alejandro Calleja†, Héctor D. Menéndez‡, Juan Tapiador† and David Camacho*
*Departamento de Informática
Universidad Autónoma de Madrid
Madrid, Spain
Email: {alejandro.martin,david.camacho}@uam.es
†Departamento de Informática
Universidad Carlos III de Madrid
Madrid, Spain
Email: {accortin,jestevez}@inf.uc3m.es
‡Department of Computer Science
University College London
London, UK
Email: h.menendez@ucl.ac.uk

***Abstract*—Android malware detection represents a current and complex problem, where black hats use different methods to infect users' devices. One of these methods consists in directly upload malicious applications to app stores, whose filters are not always successful at detecting malware, entrusting the final user the decision of whether installing or not an application. Although there exist different solutions for analysing and detecting Android malware, these systems are far from being sufficiently precise, requiring the use of third-party antivirus software which is not always simple to use and practical. In this paper, we propose a novel method called ADROIT for analysing and detecting malicious Android applications by employing meta-information available on the app store website and also in the Android Manifest. Its main objective is to provide a fast but also accurate tool able to assist users to avoid their devices to become infected without even requiring to install the application to perform the analysis. The method is mainly based on a text mining process that is used to extract significant information from meta-data, that later is used to build efficient and highly accurate classifiers. The results delivered by the experiments performed prove the reliability of ADROIT, showing that it is capable of classifying malicious applications with 93.67% accuracy.**

I. Introduction

Malware threats are present in almost all software architectures from devices firmware to super-computers. Smart devices are not an exception. Currently, the growing market of mobile apps is opening a new spreading door where black hats are able to steal critical information from users, attacking their all-day devices. Android is now the main target for these attacks, and its markets have become a sensitive bridge where black hats aim to introduce their malicious apps to get control over users mobiles. The security of these markets is critical, in order to avoid global infections and to guarantee the devices safeness.

Unfortunately, current markets, excluding Google Play Store, are performing low control services to avoid infections, leaving that responsibility to the final user [1], [2]. The user has to decide, based on the information provided by the online application store, whether they have confidence to install the app or not. Although there are different tools which can be used, mainly antivirus software, to analyse and extract information from applications and classify them between benign or malicious, these solutions are not prompt nor fully effective. They force the user to install a very resource intensive application in their devices to analyse the suspicious applications, which has also to be installed in order to perform the analysis. Considering this scenario, this work aims to provide a fast and accurate tool for detecting Android malware leveraging meta-information which can be extracted directly from the online app store and also in the Android Manifest. We combine and apply text mining and classification techniques on this information to assist the user in the decision, whether the application is reliable and it can be installed safely or when there are malicious indications which advise against installing.

Based on Gorla et al. [3] work, ADROIT aims towards providing an up-to-date and improved contribution, limiting the analysis to meta-information instead of performing software analysis, as the former authors did (which complicates and slows the analysis). This allows to build an easy-to-use technology while ensuring high accuracy, due to a wide and representative selection of features. This methodology can also help to feed recommender systems to assist the user during the decision process. Furthermore, we have also chosen supervised learning for this approach in order to improve the classification quality.

The main contributions of this paper are:

- An up-to-day analysis of Aptoide market, one of the most important alternative applications stores under which several markets are grouped, choosing randomly a representative sample of more than 9,000 apps to measure the security for the final user.
- A text mining analysis combining different meta-data

978-1-5090-4240-1/16/$31.00 ©2016 IEEE

information to generate a classifier which can assist users in their final decisions. The classifier obtains 93.67% of accuracy improving the state-of-the-art techniques and overcoming commercial Anti-Virus engines, as we demonstrate in the experiments sections.
- A technique that can easily be extended to any app market, including iOS and Windows markets, due to the independence from the application software.

The paper is structured as follows: next section explains the methodologies applied; after, Section 3 introduces the experimental setup which is applied for the experiments of Section 4. Then, next section introduces the related work. Finally, the last section details the conclusions and future work.

II. Android malware detection using meta-information

This section describes ADROIT, a novel malware detection method based on meta-information available in the app store and in the Android Manifest file. The objectives of ADROIT consist on providing a fast and reliable tool to help users when they make the decision of installing an application in their personal devices, discouraging this action when there are signs which call into doubt its real intentions. ADROIT is built in a step by a step approach as it is shown in Figure 1.

First of all, a representative set of samples was gathered with applications downloaded from Aptoide, an online app store. All these samples were analysed with a pool of different antivirus engines, allowing to verify the existence of malware in this store and generating a labelled dataset of benign and malicious Android applications. Together with the application itself, it was also downloaded diverse meta-information, later used to train and test different machine learning classification algorithms. This information includes the minimum SDK and OpenGL versions required to run the application, the minimum size of the screen and the supported CPU types of the Android device, an identification of the developer, its organisation, the locality and the country, the rating score given by the users, the number of users who voted and a description, where the developer use to describe what is the application about and define its purposes. Although it is possible to retrieve a list of permissions from the app store, this list is not reliable. After comparing it with the permissions written in the Android Manifest, it was discovered that many virus writers provide a fake list to the app store with the aim of confusing the user, as it is explained in the Experimental Setup section.

The method proposed performs a static analysis using all this meta-information with the ultimate goal of providing a fast tool to detect if an application is suspected to contain malicious code. Each piece of meta-information provides a valuable instrument to detect evidences, patterns or traces linked to apps considered as malware. For example, the rating score allows to make a decision based on the opinion of users who have already used the application. It is also useful the description, in order to discover techniques used by virus writers to confuse the user or other patterns that may reflect differences in comparison with benign-ware. The developer, organisation, locality or the country allows to find relations which can be used to disclose sectors closely linked to a particular kind of applications. Finally, the list of permissions allows to incorporate a useful definition of the behaviour of the application, enabling to find groups or relations mostly associated to a specific label.

A. Natural Language Processing

The description of the app needs special care, because it contains semantic information. In order to include the meta-information provided by the description inside our classification model we have used Text Mining techniques. The technique followed is based on the classical term-document matrix where each description is considered as a document and a set of terms is extracted from the whole description corpus. These terms are chosen after a cleaning process which eliminates those terms that are not relevant, and uniforms those terms that have similar roots. This process is divided in the following steps:

- *Remove special characters*: All irrelevant characters (such as numbers, exclamation, symbols, etc), are eliminated.
- *Remove stopwords*: those words which are normally used to connect sentences or words.
- *Stemming*: Reduces words to the word stem, i.e., to a root form where affixes can be attached.
- *Strip white spaces*: if it is found an space inside a word after applying the previous steps, it is also removed.

After the cleaning process the corpus is transformed into a term-document matrix, and normalised using TF-IDF normalisation. In addition, those terms with a sparsity above a specific threshold are removed.

Finally, this matrix is combined with the rest of meta-information available in the online app store, considering them as categorical data and translating it to a boolean matrix (excepted rating, number of votes and minimum OpenGL and SDK versions, which are kept as numeric values).

This matrix is also combined with the list of permissions defined in the Android Manifest, where each new features indicates the declaration or not of a specific permission. The use of this information allows to include more accurate data which has not been modified to confuse the user, in contrast to the meta-information available in the app store.

Once it is defined the search space, it is possible to apply different classifiers to distinguish between malware and benign-ware. The following section describes the classifiers.

B. Classification

The classification process is based on discriminating malware and benign-ware using a learning process which aims to identify patterns from different permission policies and instructions used. In this case, we consider two labels: malware and benign-ware. This information feeds the classifier in order

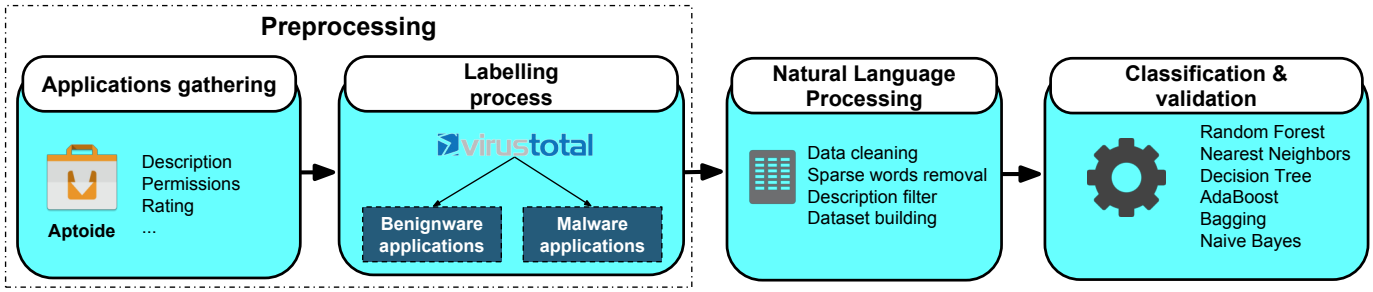

Fig. 1: Diagram of the step-by-step process to build a classifier in ADROIT

to learn the main differences among them. The following classifiers have been selected in order to perform this task[1]:

- **Random Forest (RF)**: It is a hybrid method that incorporates the advantages of combining different tree classifiers. This methodology trains several decision trees and assigns a confidence value to each one, creating a voting system. This confidence value is used to reach an agreement between the different tree classifiers [4]. It helps to determine when there are sections that are not totally linear.
- **k-Nearest Neighbours (kNN)**: The Nearest Neighbours algorithm assign a new class to an instance according to the k closest neighbours.
- **Decision Trees (DT)**: It divides the data linearly using limits in the attributes and generates a decision tree. The division is chosen using a metric, in this case, the data entropy [5]. The specific algorithm used is SCART, a modification of the C4.5 algorithm.
- **AdaBoost (AB)** It is a multi-learners approach that combines weak classifiers through a weight voting system, in order to ensure that they can complement each other. The weights are learned by the algorithm during the training process.
- **Bagging (B)** This methodology combines different classifiers choosing random subsets from the space and feeding the classifiers with them. The chosen classifiers are Decision Trees classifiers.
- **Naïve bayes (NB)**: Naïve Bayes (NB) is based on Bayes Probability Laws and considers each feature independently from the rest [6] and contributing to the information modelling. This helps to understand when the features are independent or they need to complement their information with another feature.

III. Experimental Setup

This section introduces the data and algorithms that have been applied for this study, specially the parameters and validation metrics used during the experiments.

[1]The implementation of the classifiers can be found in http://scikit-learn.org/stable/

A. Dataset

Currently, there are several app stores enabling users to download applications to their devices. Although they claim that every available application has passed through a full scan to detect if it contains malicious components, it has been proved that many applications are able to bypass these filters and finish being published. We have gathered a set of 12,360 applications downloaded from the Aptoide[2] app store. Then, we analysed each of them using the API provided by the VirusTotal[3] online portal, which allows to analyse applications or documents with 56 different antivirus engines. For more than the 38% of the samples, at least one antivirus tested for positive. This figure notes the need to take precautions when downloading and installing new applications, even when they are obtained from sites qualified as safe.

These 12,360 apps extracted from the Aptoide market[4] were used to test our model. In order to generate a ground truth about the software nature, the results obtained from the VirusTotal website were used to label all the samples. We considered that an app is malware when a single Anti-Virus produces a positive detection. From all these application, VirusTotal detected 4,799 apps as malware, while the rest of samples were categorised as benign-ware by all the Anti-virus engines[5].

Together with the application executable file, we took all the meta-information information previously described, including the description, the developer or the rating score. After an overall analysis, it was found that an important number of the samples did not have a description, forcing to remove all these applications from the dataset and decreasing its size to 2,426 malicious and 6,704 benign applications.

B. Text Mining

The text preprocessing step for each application aims to obtain a set of relevant terms that can be used to represent them and distinguish between malicious versus benign applications. This Text Mining process was addressed using the Text Mining Package for R[6]. Once the data is preprocessed and cleaned

[2]https://www.aptoide.com/
[3]https://www.virustotal.com/
[4]http://www.aptoide.com/
[5]The extraction and analysis date was done on June 2016.
[6]https://cran.r-project.org/web/packages/tm

	Malware	Benignware
Samples where permissions informed in website differs from Manifest	73.32%	37.03%
Average number of permissions informed in app store website but not declared in Manifest	7.58	2.25
Average number of permissions informed in Manifest not declared in app store	9.90	9.63

TABLE I: Differences in Android permissions declaration found in Android Manifest and app store

applying different filters, it is needed to find a subset of words representative of different behavioural patterns. The RemoveSparse function allows to remove those terms which are only part of a low number of samples and hardly to be useful to discover these patterns. This function receives a parameter in charge of establishing a limit to dispersed terms included. Given the importance of this parameter, which is closely linked to the granularity level of the system, but also to its generalisation ability on new data, different values ranging from 0.95 to 0.999 were tested.

C. Android permissions

When developing an Android application, it is needed to indicate in the Android Manifest a series of permissions required to run the app. They allow the application to access certain operating system and terminal functions and are a means of increasing the security. They also enable to extract an overall picture of its behavioural model based on the actions expected to be performed at run-time. These permissions are provided to the user in the app store website, as well as other information such as the minimum required SDK version or the description of the application. Although the information displayed should be consistent with the Android Manifest, this is not always the case.

In order to analyse and discover possible differences between both information sources, the app store website and the Android manifest, they were compared separately for all the benign and malicious applications gathered from Aptoide. Table I summarises the results obtained. There are clear differences between malware and benign-ware. While 37% of the second ones showed differences between the permissions declared in the Manifest and in the app store, in the case of malware this figure rises to 73% of the samples. It is also worth noting the specific distribution between the differences found in malware and benign samples, where there can be permissions declared in the Android Manifest, but not in the app store website or, in contrast, permissions declared in the app store but missing in the Manifest. Regarding the first group, there is a very remarkable trend, showing that virus-writers use to report many permissions which are later not implemented. The reasons for this could lie in the intention of black-hats of showing fictitious intentions to hide the real purposes (as a Trojan horse).

On the other hand, within the range of applications reporting an incomplete list of permissions to the app store, there is a similar trend for both kinds of applications, where close to 9.5% of the samples exhibit discrepancies. Although this is an unexpected behaviour, it can be attributed to an intentional omission of particular permissions by the app store. Due to this divergence between the permissions declared in the Android Manifest and on the app store website, we have selected the Manifest as a more reliable source of information, since it is the document actually used by the operating system to allow the application execution.

D. Classification

The classification algorithms applied in this study were executed using the scikit-learn[7] library for Python. Each classification algorithm was executed 20 times with a different random initialisation for each configuration evaluated. In order to measure the quality of each model, the data was divided into a training and a testing dataset, at a rate of 2/3 and 1/3 respectively. The specific classification algorithms used are the following:

- **Random Forest:** fixing the number of internal decision trees to 100.
- **Nearest Neighbors:** keeping default settings.
- **Decision Tree:** keeping default settings.
- **AdaBoost:** keeping default settings.
- **Naive Bayes:** keeping default settings.
- **Bagging:** using Random Forest as individual classifier and fixing the number of decision trees to 100.

In order to evaluate the quality of the algorithms, we have used the accuracy:

$$ACC = \frac{TP + TN}{TP + TN + FP + FN}, \tag{1}$$

which allow us to compare the current results with Gorla et al. results [3]. Furthermore, we have also built the ROC curve for different configurations in order to assess the false positives and true positives ratios.

IV. Experiments

This section shows the final experiments performed during the detection process. First, we focus the analysis on the classification accuracy. After, we perform an study of the ROC curve in order to evaluate what classifier can be more suitable to reduce false positives. Finally, we compare our detection methodology with commercial antivirus and another method with a similar approach.

A. Accuracy Results

Each classification algorithm run 20 times for different number of terms used in the text-mining step (according to the parameter of the RemoveSparse function). The results are shown in Fig. 2, where Naive Bayes and Nearest Neighbours have been omitted for better visualisation, since any of them exceeded a 80% accuracy at no observed point. The remaining four classifiers also indicate important differences between them. While RandomForest and Bagging have a very similar behaviour, a simple decision tree or AdaBoost are not able to compete on accuracy.

[7] http://scikit-learn.org/

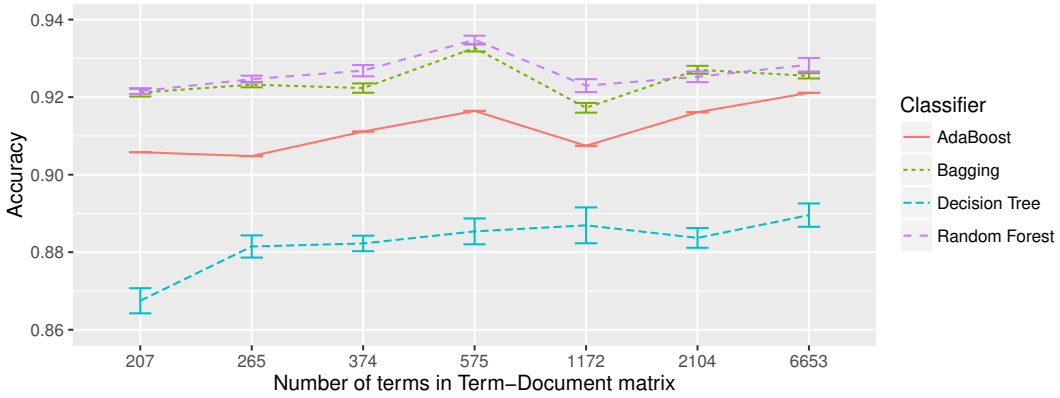

Fig. 2: Accuracy reached by the different classifiers trained according to the number of terms used in the text-mining process

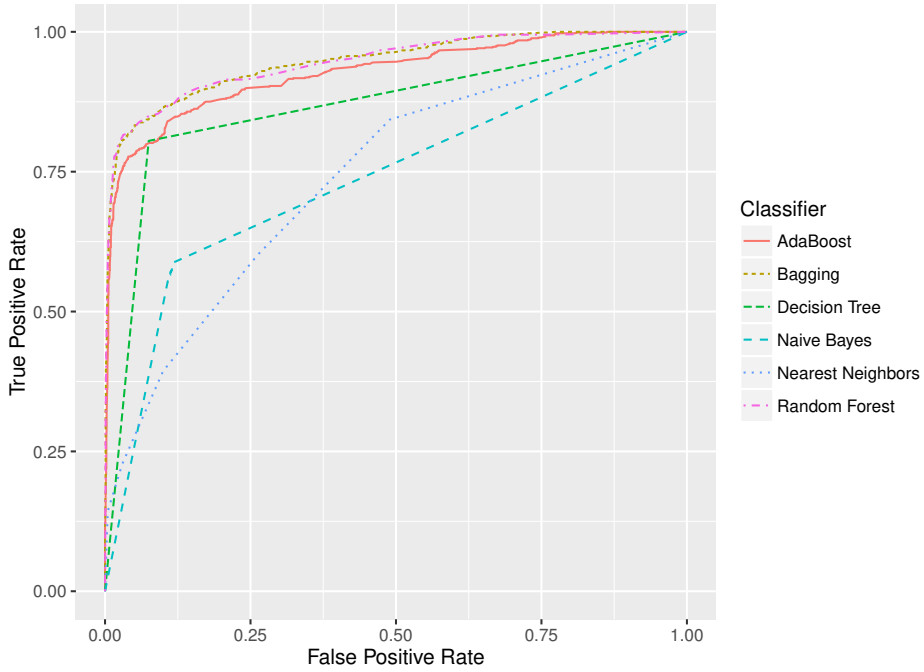

Fig. 3: ROC curve for the different classification algorithms tested

Comparing RandomForest with a Bagging algorithm, the former is able to slightly increase the accuracy for most of the sizes of the term-document matrix tested. In terms of deviation, Decision Tree produces very different solutions, while Random Forest and Bagging produce solutions closed to the average. The best result, a 93.67% accuracy, is achieved using a Random Forest classifier and 575 terms (applying a parameter of 0.98 in the remove sparse function).

The number of terms used to include the information provided by the description of the application, 575 different words, can be considered as a large figure, denoting that it is needed a deep enough granularity level to benefit from this information. However, a larger number of points contributes negatively, by entering unnecessary data to the model and causing accuracy to drop slightly. In general, the accuracy level achieved can be considered as high, arising close to 94% accuracy without requiring a dynamic analysis and just using information retrieved from the app store website and from the Android Manifest.

B. ROC study

While the accuracy is a reliable information source to measure the quality of a classifier, the false positive rate is also a key when designing malware detection methods, because of its link with the proper system operation. Fig. 3 shows the ROC curve for the six classification algorithms tested. Once again, Naive Bayes and Nearest Neighbours yield the worst results with the smaller AUC (Area Under the Curve). Random Forest and Bagging produce almost identical results, with 94% and 94.04% AUC sizes respectively. As it can be seen, although Random Forest is able to achieve a higher accuracy

Detection method	Accuracy
ADROIT	**93.67%**
CHABADA	81.18%
Qihoo-360	85.30%
Alibaba	85.06%
ESET-NOD32	74.92%
Fortinet	73.01%
Cyren	72.79%
NANO-Antivirus	72.40%

TABLE III: Accuracy reached with different commercial antivirus engines, the CHABADA method and the ADROIT

level, Bagging produces a larger Area Under the Curve or, in other words, a better balance between true positives and false positives rates.

For best understanding of the ROC curves generated by the different classifiers, Table II shows the maximum true positive ratio achieved (depending on the cut-off value used) according to different false positive rates accepted. With the aim of minimising the false positive rate, Bagging is the best option, reaching an 86% accuracy with only 10% of False Positives. In contrast, when the objective is to reach the highest true positive rate, it is possible to arise 100 per cent using a Decision Tree classifier while producing 50% of False Positives. These values show that it is possible to configure the algorithm to give greater emphasis to a specific parameter. The final decision will depend on the user's decision.

C. Results comparison

ADROIT aims to provide a fast tool to analyse and determine the nature of an Android applications without compromising the accuracy. The meta-information, the permissions, as well as the text mining step, allow to build a strong classifier able to compete and to overcome commercial antivirus engines and also CHABADA, a similar approach to ADROIT but where the authors use clustering techniques. The results, displayed in Table III, show that ADROIT is able to overcome the accuracy levels achieved by the rest of methods tested by more than 8%.

Based on these results, in can be asserted that the meta-information accompanying each application offers a powerful resource to train accurate classification models able to distinguish between malicious and benign applications. This means that the description or a permission policy specific definition provide significant details about the nature of an Android application.

V. Related Work

This section introduces the related work based on three main points of view which are combined in this work: Android apps analysis, malware analysis and text mining techniques.

a) Android analysis: Android has become a famous OS for smart devices over the last few years. This OS covers from mobile phones, to tablets or to TV devices among others. There are several research lines focused on Android app analysis, specially on understanding the market behaviour [7], development decision [8], app testing [9] and prediction models to detect markets tendencies [10]. The work developed by Gorla et al. [3], which inspires this work, was focused on detecting anomalies in the market, based on the different categories where the apps can be distributed. Using this anomaly detector, authors where able to detect malware, but with a high false positive ratio. This work aims to continue this strategy focusing only on those features that are easy to extract, instead of performing an API calls analysis, which is normally not available to the final users.

b) Malware analysis: The analysis of Android malware is contextualised in the malware Arms Race. From the beginning of computer science, malware has been developed in order to attack system vulnerabilities with a wide range of malicious goals in mind [11]. These harmful attacks, originally promoted by single individuals and currently promoted by strong organisations, are usually counteract by white hats, which are usually single individuals, researchers or small organisations [12].

There are several ways for detecting malware, which are categorised from a software analysis point of view as dynamic and static analysis.

On the one hand, static analysis aims to analyse malware using only the information provided by the program itself [13]. This is usually performed by a disassembly or decompilation process which recovers the program instructions and the Control Flow Graph. Using this information, different strategies, such as symbolic execution [14], opcode analysis [15] or control flow graph analysis [16] can be used to detect and understand malware behaviour. In Android systems, we can find static analysis examples like CHABADA [3] which extract API calls from the apps, the work of Schmidt et al. [13] which define a collaborative framework to share static analysis features between different systems, MOCDroid [17], which uses genetic algorithms or the work of Yerime et al. [18], where authors combine static features with Bayesian classifiers.

On the other hand, dynamic analysis is focused on malware execution and aims to understand its behaviour, studying the traces it leaves in the systems [19], such as memory access [20], network communications [21], and registers modifications [22], among others. This information creates a signature used for the detection and discrimination processes. Some examples of dynamic analysis in Android environments are CopperDroid [23], which reconstructs the behaviour of an app, DREBIN [24], which combines dynamic features with classification and ALTERDROID [25], which detects obfuscated components from behavioural traces.

In this work we only focus on the meta-information recorded by the market, enabling to make a fast but also detailed analysis through the large number of features available in the app store, such as the developer identification or a description of the application. All this information is processed using text mining and machine learning classifiers to develop a system able to categorise Android applications between benign and malicious depending on the existence of certain patterns. While most of the features employed to train a

Classifier \ False Positive rate	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
Bagging	0	0.86	0.89	0.91	0.94	0.96	0.98	0.99	1	1	1
Random Forest	0	0.85	0.88	0.91	0.93	0.95	0.97	0.99	0.99	0.99	1
Decision Tree	0	0.81	0.81	0.81	0.81	1	1	1	1	1	1
Naive Bayes	0	0.73	0.75	0.82	0.87	0.89	0.89	0.89	1	1	1
AdaBoost	0	0.55	0.87	0.90	0.92	0.95	0.97	0.98	0.99	1	1
Nearest Neighbours	0	0.38	0.60	0.65	0.70	0.80	0.83	0.84	1	1	1

TABLE II: True Positive ratios according to different False Positive ratios

classification algorithm can be used as polynomial attributes, the description of an application is formed by texts of different length providing raw information. The use of Text Mining techniques allows to extract relevant data from these texts.

c) Text Mining: Text Mining is one of the most relevant subfields of Natural Language Processing [26] specially focused on topic detection and extraction [26], sentiment analysis [26] and recommender systems [27], among others. These techniques are based on finding similarities among documents and topics based on semantic features extracted from their used [27]. Using these features as basic knowledge, machine learning algorithms leverage them to generate high-level abstraction models [27]. These models can be applied to document classification [27], trending detection [26], etc. In this work we take advantage of this combination to generate a detection system for malicious apps, mainly based on understanding the descriptions language and the app meta-information.

VI. Conclusions and Future Work

This work has presented a text mining and machine learning approach to detect malicious apps in a well-known Android market, Aptoide. The combination of these two techniques is applied over the meta-information extracted from the market website and the Android Manifest, which includes developer data, the permissions or the description of the application, where a text mining operation is in charge of extracting relevant information. The final model, called ADROIT, use all this information to train a machine learning classification algorithm able to reach accuracy rates close to 94%. One of the strengths and contributions of this paper is related to the employment of simple text mining techniques that allows to generate highly accurate results. Our future work involves applying and testing ADROIT in other app stores, as it can be Google Play Store, increasing the accuracy by crossing information of the same application in different app stores or incorporating further meta-information.

Acknowledgment

This work has been supported by the next research projects: Spanish Ministry of Economy and Competitivity and European Regional Development Fund FEDER (TIN2014-56494-C4-4-P), MINECO grant TIN2013- 46469-R (SPINY: Security and Privacy in the Internet of You), the CAM grant S2013/ICE-3095 (CIBERDINE: Cybersecurity, Data, and Risks) and SeMaMatch EP/K032623/1.

References

[1] K. Chen, P. Wang, Y. Lee, X. Wang, N. Zhang, H. Huang, W. Zou, and P. Liu, "Finding unknown malice in 10 seconds: Mass vetting for new threats at the google-play scale," in *24th USENIX Security Symposium (USENIX Security 15)*, 2015, pp. 659–674.

[2] T. Isohara, K. Takemori, and A. Kubota, "Kernel-based behavior analysis for android malware detection," in *Computational Intelligence and Security (CIS), 2011 Seventh International Conference on*. IEEE, 2011, pp. 1011–1015.

[3] A. Gorla, I. Tavecchia, F. Gross, and A. Zeller, "Checking app behavior against app descriptions," in *Proceedings of the 36th International Conference on Software Engineering*. ACM, 2014, pp. 1025–1035.

[4] T. K. Ho, "The random subspace method for constructing decision forests," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 20, no. 8, pp. 832–844, 1998.

[5] J. R. Quinlan and R. L. Rivest, "Inferring decision trees using the minimum description lenght principle," *Information and computation*, vol. 80, no. 3, pp. 227–248, 1989.

[6] P. Domingos and M. Pazzani, "On the optimality of the simple bayesian classifier under zero-one loss," *Machine learning*, vol. 29, no. 2-3, pp. 103–130, 1997.

[7] A. A. Al-Subaihin, F. Sarro, S. Black, L. Capra, M. Harman, Y. Jia, and Y. Zhang, "Clustering mobile apps based on mined textual descriptions," in *Proceedings of the 36th International Conference on Software Engineering*. ACM, 2016.

[8] F. Sarro, A. A. Al-Subaihin, M. Harman, Y. Jia, W. Martin, and Y. Zhang, "Feature lifecycles as they spread, migrate, remain, and die in app stores," in *2015 IEEE 23rd International Requirements Engineering Conference (RE)*. IEEE, 2015, pp. 76–85.

[9] K. Mao, M. Harman, and Y. Jia, "Sapienz: Multi-objective automated testing for Android applications," in *Proc. of ISSTA'16*, 2016, to appear.

[10] W. Martin, M. Harman, Y. Jia, F. Sarro, and Y. Zhang, "The app sampling problem for app store mining," in *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*. IEEE, 2015, pp. 123–133.

[11] P. Szor, *The art of computer virus research and defense*. Pearson Education, 2005.

[12] S. Bratus, "What hackers learn that the rest of us dont," *IEEE Security and Privacy*, 2007.

[13] A.-D. Schmidt, R. Bye, H.-G. Schmidt, J. Clausen, O. Kiraz, K. A. Yuksel, S. A. Camtepe, and S. Albayrak, "Static analysis of executables for collaborative malware detection on android," in *2009 IEEE International Conference on Communications*. IEEE, 2009, pp. 1–5.

[14] D. A. Ramos and D. Engler, "Under-constrained symbolic execution: correctness checking for real code," in *24th USENIX Security Symposium (USENIX Security 15)*, 2015, pp. 49–64.

[15] I. Santos, F. Brezo, J. Nieves, Y. K. Penya, B. Sanz, C. Laorden, and P. G. Bringas, "Idea: Opcode-sequence-based malware detection," in *International Symposium on Engineering Secure Software and Systems*. Springer, 2010, pp. 35–43.

[16] D. Bruschi, L. Martignoni, and M. Monga, "Detecting self-mutating malware using control-flow graph matching," in *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, 2006, pp. 129–143.

[17] A. Martín, H. D. Menéndez, and D. Camacho, "Mocdroid: multi-objective evolutionary classifier for android malware detection," *Soft Computing*, pp. 1–11, 2016.

[18] S. Y. 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*. IEEE, 2013, pp. 121–128.

[19] C. Willems, T. Holz, and F. Freiling, "Toward automated dynamic malware analysis using cwsandbox," *IEEE Security and Privacy*, vol. 5, no. 2, pp. 32–39, 2007.

[20] A. Dinaburg, P. Royal, M. Sharif, and W. Lee, "Ether: malware analysis via hardware virtualization extensions," in *Proceedings of the 15th ACM conference on Computer and communications security*. ACM, 2008, pp. 51–62.

[21] R. Perdisci, W. Lee, and N. Feamster, "Behavioral clustering of http-based malware and signature generation using malicious network traces." in *NSDI*, 2010, pp. 391–404.

[22] A. Vasudevan and R. Yerraballi, "Cobra: Fine-grained malware analysis using stealth localized-executions," in *2006 IEEE Symposium on Security and Privacy (S&P'06)*. IEEE, 2006, pp. 15–pp.

[23] K. Tam, S. J. Khan, A. Fattori, and L. Cavallaro, "Copperdroid: Automatic reconstruction of android malware behaviors," in *Proc. of the Symposium on Network and Distributed System Security (NDSS)*, 2015.

[24] D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, and K. Rieck, "Drebin: Effective and explainable detection of android malware in your pocket." in *NDSS*, 2014.

[25] G. Suarez-Tangil, J. E. Tapiador, F. Lombardi, and R. Di Pietro, "Alterdroid: Differential fault analysis of obfuscated smartphone malware," *IEEE Transactions on Mobile Computing*, vol. 15, no. 4, pp. 789–802, 2016.

[26] B. Pang and L. Lee, "Opinion mining and sentiment analysis," *Foundations and trends in information retrieval*, vol. 2, no. 1-2, pp. 1–135, 2008.

[27] R. Baeza-Yates, B. Ribeiro-Neto *et al.*, *Modern information retrieval*. ACM press New York, 1999, vol. 463.