The evolution of permission as feature for Android Malware detection

José Gaviria de la Puerta, Borja Sanz, Igor Santos Grueiro and Pablo García Bringas

DeustoTech Computing, University of Deusto
jgaviria@deusto.es, borja.sanz@deusto.es, isantos@deusto.es,
pablo.garcia.bringas@deusto.es

Abstract. Over the last few years, the presence of mobile devices in our lives has increased, offering us almost the same functionality as personal computers. Since the arrival of Android devices, the amount of applications available for this operating system has increased exponentially. Android has become one of the most popular operating systems in these devices. In fact, malware writers insert malicious applications into Android using the Play store and other alternative markets. Lately, many new approaches have been made. Sanz et al, for instance, presented PUMA, a method used to detect malicious apps just by taking a look at the permissions. In this paper, we present the differences between that interesting approach and a newer and bigger dataset. Besides, we also present an evolution in the permissions along the years.
Keywords: Android, Malware, Permissions, Detection, Machine Learning.

1 Introduction

Nowadays, smartphones are becoming increasingly popular. These small computers come with us everywhere, allowing us to check our email, browse the Internet or play games with our friends. However, in order to be able to use all the possibilities that these devices offer, we need to install some applications in our smartphones.

When the first smartphones appeared, some users had lots of problems when installing those apps, as it was really difficult to find a centralized place where they could obtain them. That is why, if they wanted to do that, the only alternative they had was to search for them on the Internet.

When the users finally found the application they wanted to install, problems began. In order to protect the device and avoid piracy, several operating systems, such as Symbian, employed an authentication system based on certificates that caused several inconveniences for the users (e.g., they could not install applications even if they had bought them).

Nowadays, there are many new methods to distribute these applications. Thanks to the deployment of Internet connections in mobile devices, users can

install any application without even connecting the mobile device to the computer. Apple's AppStore[1] was the first store to implement this new model and was so successful that other manufacturers, such as Google, RIM and Microsoft, have followed the same business model developing application stores accessible from the device. Now, users just need to create an account for an application store to buy and install new applications.

This new way of selling apps in a market created a new factor for the community of developers. It was a new world for them, so they started to create apps for every existing mobile platform. Furthermore, those factors have drawn developers' attention to these platforms. According to Statista[2], the number of available applications on the App Store is over 1,200,000, whereas the Play Store of Android, which is the operating system of Google, has over 1,300,000. Figure 1 represents all the apps by operating system in July 2014.

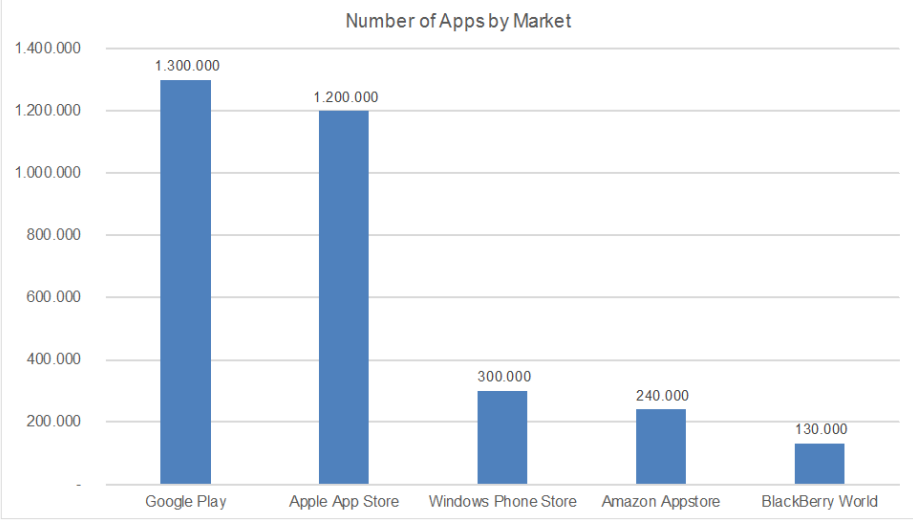

Fig. 1. Mobile apps by operating system in July 2014.

In the same way, malicious software has arrived to both platforms. There are several applications whose behaviour is, at least, suspicious of trying to harm the users. There are other applications that are definitively malware. That is the reason why, those platforms have used different approaches to protect against this type of software.

According to their response to the US Federal Communication Commission's July 2009[3], Apple applies a rigorous review process made by at least two re-

[1] http://www.apple.com/iphone/features/app-store.html
[2] http://www.statista.com
[3] http://online.wsj.com/public/resources/documents/wsj-2009-0731-FCCApple.pdf

viewers. In contrast, Android relies on its security permission system and on the user's sound judgment. Unfortunately, users have usually no security consciousness and they do not read the required permissions before installing an application. This is one of the most important things that the security teams want the users to learn.

In spite of the fact that both AppStore and Play Store include clauses in the terms of services that urge developers not to submit malicious software, both have hosted malware in their stores. One of the solutions that these enterprises have developed, is to remove remotely the malicious applications that are installed in the devices. Sadly, the usage of these models is insufficient to ensure the user's security, and that is why the new models have been included.

It is demonstrated that the usage of machine learning techniques for generic malware detection and classification is widely applied in the scientific community[?]. Besides, several approaches [?,?] have been proposed to classify applications specifying the malware class; e.g., trojan, worms, virus; and even the malware family.

Section 2 explains the permission security system that is used by Android. In section 3 the both dataset, malware and goodware, are exposed. We can also see how we obtained the goodware dataset.

The next section is the evolution of the permissions. In this section we can see a graph with the difference between permissions in different times.

Section 4 shows us all the experimentation of the paper. Here, all the algorithms that we use in the approach are included. Moreover, the results and the discussion are included in it. Section 5 presents the conclusions of the paper.

2 Permission in Android

An additional security system that Android provides the operating system is the "permission" mechanism. With it, the S.O. enforces the restrictions of each app in the mobile device.

The normal functioning of these permissions is quite simple: for example, if one app wants to use the connectivity through Internet, it has to request the permission associated with the connection to it. These permissions are usually given at installation time, being the user the person that has to take the decision of installing the app after reading and understanding the permissions it requests.

Fig. 2. Permissions in a *Manifest.xml* file for one application.

Figure 2 is an example of a Manifest.xml with different permissions for an application. All the permissions must appear in the clause entitled "uses-

permission". If an app wants to access the Internet and does not have the "android.permission.INTERNET ", it will not be able to do it.

3 Information gathering

In this paper, we have chosen to compare two different datasets to see their evolution over the last few years in terms of permissions selected by the apps.

3.1 Older dataset

The first dataset is the one that was selected by Sanz et al. in their paper called "PUMA: Permission Usage to Detect Malware in Android" [?]. This dataset is composed by a total of 357 applications of goodware, and 249 apps of malicious software. Malware samples were gathered by means of *VirusTotal*[4], which is an analysis tool for suspicious files. We have used their service called VirusTotal Malware Intelligence Services, which is available for researchers to perform queries to their database.

With the goal of developing the goodware dataset, Sanz et al. gathered a collection of 1811 Android applications of different types. In order to classify them properly, they chose to follow the same naming as the official Android market. To this end, they used an unofficial Android Market API[5] to connect with the Android market and, therefore, obtain the classification of the applications.

They selected the number of applications within each category according to their proportions in the *Play Store*, before it was called *Android Market*. Despite of Android has got some types of applications, they did not make distinctions to choose one, or more, of these types. So, in that dataset all types are represented. Then, they selected randomly the apps from different categories.

3.2 New dataset

The second dataset has been generated using two different techniques. Goodware samples have been gathered using the Selenium[6] application for automating web browser. With this automation we use the application web APIfy[7] to download Android applications from different categories.

Totally, we have downloaded 7.062 applications from Google Play. These apps have been gathered from different available categories in that market. That is exactly what we see in Table 1.

The choice of these lists for obtaining samples was completely random, as what we wanted was to have a heterogeneous applications dataset from Google's store. To verify that the samples did not contain malicious code, we used the

[4] http://www.virustotal.com
[5] http://code.google.com/p/android-market-api/
[6] http://docs.seleniumhq.org/
[7] http://apify.ifc0nfig.com/

Table 1. Number of App by category

Category	*Number*	*Category*	*Number*
BUSINESS	179	GAME ACTION	156
LIFESTYLE	168	GAME PUZZLE	170
SHOPPING	163	GAME SIMULATION	165
BOOKS AND REFERENCE	173	ENTERTAINMENT	157
MEDICAL	176	GAME TRIVIA	161
GAME STRATEGY	163	GAME ROLE PLAYING	159
GAME ADVENTURE	161	NEWS AND MAGAZINES	170
GAME CASUAL	160	FINANCE	173
MEDIA AND VIDEO	156	GAME WORD	170
GAME CARD	176	APP WALLPAPER	163
LIBRARIES AND DEMO	185	PRODUCTIVITY	178
WEATHER	172	MUSIC AND AUDIO	156
GAME EDUCATIONAL	170	COMMUNICATION	177
GAME BOARD	175	HEALTH AND FITNESS	172
EDUCATION	179	COMICS	61
GAME FAMILY	157	GAME ARCADE	169
GAME SPORTS	168	GAME RACING	168
SOCIAL	163	TRAVEL AND LOCAL	174
GAME CASINO	175	PHOTOGRAPHY	159
TRANSPORTATION	174	TOOLS	177
SPORTS	161	PERSONALIZATION	111
GAME MUSIC	162		

Total: 7.062

online platform VirusTotal. This platform uses 43 antivirus engines to scan the sample that has been sent previously. This analysis returns the total number of engines that have been detected as malicious and malware is for that engine. Since the experiment was desired to have all the possible clean malware samples, we made the decision that if a sample was detected as malicious by at least one of the antivirus engines, it would be immediately separated from the rest of the dataset. Furthermore, the ones that were detected as adware[8], were also separated from that dataset right away.

After finishing that analysis, we chose a total of 5.511 goodware apps. This result implied that 21.96% of the downloaded apps were considered malware by at least one antivirus engine.

The malware dataset was gathered from the Drebin malware dataset [?], which is composed by a total of 5,560 samples of malware and which is divided into different families.

[8] Adware is a type of action hidden in applications, which send targeted advertisements to our device when you run an application

3.3 Permissions through the years

Previously, it has been mentioned what a manifest file is. In it, we can see all the permissions that an application requests to the user. Android, for instance, has got a total of 150 permissions defined in its API.

Despite of this, some companies, like, for example, Samsung, have made their own group of permissions for using their terminal. In this experiment we do not study this type of permissions, we only look at the permissions of the Android Platform.

Figure 3 shows that there is one permission that every app in the market requests: "Internet". Besides, we can also see that, nowadays, the number of permissions required by many malware has increased greatly over the last few years, something that indicates that malware is more sophisticated now.

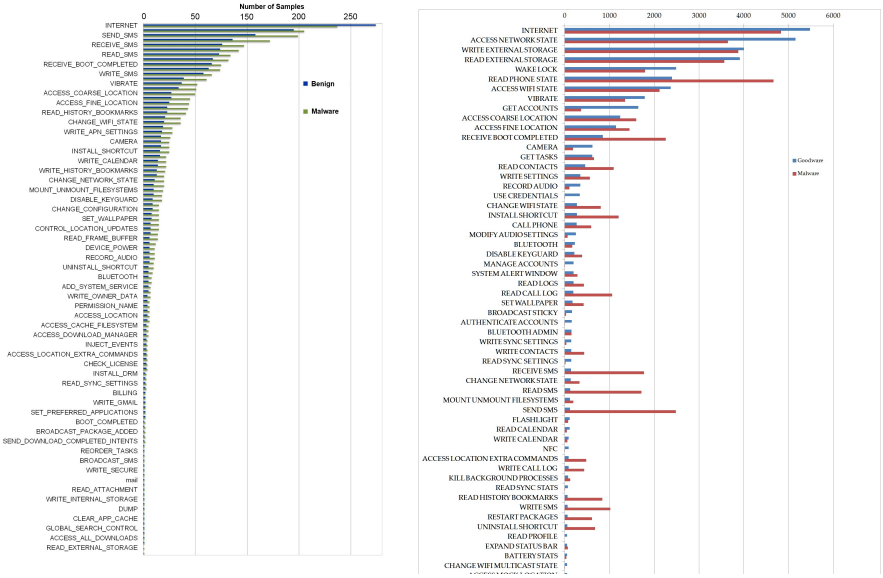

Fig. 3. Difference between permissions request in PUMA dataset and this experiment dataset.

4 Experimental validation

To compare the two datasets, we have employed supervised machine learning methods to classify Android applications into malware and benign software. To this extent, we have used Waikato Environment for Knowledge Analysis (WEKA)[9]. In particular, we used the classifiers specified in Table 2. To evaluate the performance of machine-learning classifiers, k-fold cross validation is

[9] http://www.cs.waikato.ac.nz/ml/weka/

usually used [?]. Thereby, for each classifier we tested, we performed a k-fold cross validation [?] with k = 10. In this way, our dataset was split 10 times into 10 different sets for learning (90% of the total dataset) and testing (10% of the total data).

Table 2. Algorithms that are used for classification.

Algorithms
IBK 1
IBK 3
IBK 5
SimpleLogistic
NaiveBayes
BayesNet K2
BayesNet TAN
SMO Poly
SMO NPoly
J48
RandomTree
RandomForest 10
RandomForest 50
RandomForest 100

4.1 Machine learning algorithms

In this research, we chose to compare the performance of different classification algorithms given the occasionally notable differences in effectiveness that can be observed in similar experiments conducted in other areas [?]. The algorithms used for the tests were the following: Random Forest, J48, Bayes Theorem-based algorithms, K-Nearest Neighbor (KNN), Sequential Minimal Optimization (SMO) and Simple Logistic.

- Random Forest. Random Forest is an aggregation classifier developed by Leo Breiman [?] which is formed by a bunch of decision trees considered in a way in which the introduction of a stochastic component improves the Accuracy of the classifier, either in the construction of the trees or either in the training dataset.
- J48. J48 is an open source implementation for Weka of the C4.5 algorithm [?]. C4.5 creates decision trees given an amount of training information making use of the concept *information entropy* [?]. The training data consist of a group $S = s_1, s_2, ..., s_n$ of already classified samples $s_1 = x_1, x_2, ..., x_m$ in which $x_1, x_2, ..., x_m$ represent the attributes or characteristics of each sample. In each node of the decision tree, the algorithm will choose the attribute in the data that most efficiently divides the dataset in enriched choruses of a given class using the entropy difference or the already mentioned *normalized information gain* as selective criteria.

– Bayes Theorem-based algorithms. The Bayes Theorem, the base for the Bayesian inference, is a statistical method which determines, based on a number of observations, the probability of a certain hypotheses being true. For the classification needs here exposed, this is the most important capability of Bayesian networks: in our case, the probability of an app being malicious or bening The theorem is capable of adjusting the probabilities as soon as new observations are performed. Thus, Bayesian networks conform a probabilistic model that represents a collection of randomized variables and their conditional dependencies by means of a directed graph. We have trained our models with three different search algorithms: K2, Hill Climbing and TAN.
  In this group we have also considered the inclusion of Naïve Bayes. The idea is that if the number of independent variables managed is too big, it does not make any sense to make probability tables [**?**]. Then, the reduced model with simplified datasets give to the algorithm the appellative of *Naïve*.
– K-Nearest Neighbor (KNN). The KNN algorithm is one of the most simple classification algorithms amongst all of those available for the machine learning techniques. It takes decisions based on the results of the k closest neighbours to the analysed sample in the experimental n-dimensional space ($\forall k \in \mathbb{N}$). In this case, and taken into account the simplicity of the algorithm, we have explored even more values ($k = 1, 3, 5$) so as to determine if this enlargement would throw any kind of additional advantage.
– Sequential Minimal Optimization (SMO). SMO, invented by John Platt [**?**], is an iterative algorithm used for the solution of the optimization problems that appear when training *Support Vector Machines* (SVM). Basically, SMO divides the problem into a series of smaller subproblems which are analytically solved lately.
  At this point, we have selected different kernels with these algorithms: a polynomial kernel, a normalized polynomial kernel.
– Simple Logistic. This algorithm is used to predict the result of a variable function of the independent variables, or predictor. The logistic regression formula is:

$$Y_i = \frac{1}{1 + e^{-(\beta_0 + \beta_1 X_{1,i} + ... + \beta_d X_{d,i})}} \tag{1}$$

  Being Y_i the classification to be predicted by the model, in our case it would be *goodware* or *malware*. The variable X is the vector with the extracted permissions for a specific application, finding that $X_{d,i}$ is the value assigned to an n-gram of permissions in d enforcement position that is in row i. Parameters $beta_{ast}$ are determined by the algorithm in the training phase.

4.2 Validation employed parameters

This evaluation was performed according to the following parameters, usually employed to compare the performance of different algorithms in the field of machine learning:

– *True Positive Ratio* (TPR),which is calculated by dividing the number of bening apps correctly classified (TP) between the total samples taken ($TP+FN$). The formula is:

$$TPR = \frac{TP}{(TP + FN)} \tag{2}$$

– *False Positive Ratio* (FPR),which is calculated by dividing the number of samples corresponding to malicious apps whose classification (FP) were missed by the total number of samples ($FP + TN$). The formula is:

$$FPR = \frac{FP}{(FP + TN)} \tag{3}$$

– *Accuracy* (P), which is calculated by dividing the total hits by the total number of instances in the dataset. The formula is:

$$P = \frac{TP + TN}{TP + FP + TN + FN} \tag{4}$$

– *Area Under ROC Curve* (AUC) [**?**], that establishes the relationship amongst the false negatives and the false positives. The ROC Curve it is usually used to generate statistics that represent the performance or the effectiveness in a wider sense of a classifier.

4.3 Results and discussion

Using the algorithms and the parameters above mentioned to compare the performance, the Table 3 shows us the different results recovered with both datasets. The first question that we solved was if the usage of a bigger dataset improved the classification or not. Seeing the result in Table 3, all the algorithms improve their accuracy and all their parameters. Once again, it is also demonstrated that the best algorithm is *Random Forest* but using a different number of trees.

Something curious is the values of the FPR parameter in Drebin dataset. Moreover, the good values of AUC are also important, being of 0.99 in more than one case.

Using the new dataset, all the methods achieved accuracy rates higher than 85%. The best classifier, in terms of accuracy, is *Random Forest*, with 100 trees.

Also in terms of TPR the best result that we have obtained is 0.97 in the classifiers *Random Forest* with 10, 50 and 100 trees, *SMO* with PolyKernel and Normalized PolyKernel, *Bayes Net* with TAN and *Simple Logistic*. The lowest FPR goes for *Random Forest* with 10, 50 and 100 trees and *IBK* with K=1.

5 Conclusions

Permissions are the most recognisable security features in Android. The user must accept them in order to install any application. In this paper, we validate

Table 3. Result for the classification algorithms using the PUMA dataset and the Drebin dataset

PUMA Dataset

Algorithm	TPR	FPR	AUC	Accuracy
IBK 1	0.92	0.21	0.90	85.55%
IBK 3	0.90	0.22	0.89	83.96%
IBK 5	0.87	0.24	0.88	81.91%
SimpleLogistic	0.91	0.23	0.89	84.08%
NaiveBayes	0.50	0.15	0.78	67.64%
BayesNet K2	0.45	0.11	0.77	67.07%
BayesNet TAN	0.53	0.16	0.79	68.51%
SMO Poly	0.91	0.26	0.83	82.84%
SMO NPoly	0.91	0.19	0.86	85.77%
J48	0.87	0.25	0.86	81.32%
RandomTree	0.90	0.23	0.85	83.32%
RandomForest 10	0.92	0.21	0.92	85.82%
RandomForest 50	0.91	0.19	0.92	86.41%
RandomForest 100	0.91	0.19	0.92	86.37%

DREBIN Dataset

Algorithm	TPR	FPR	AUC	Accuracy
IBK 1	0.96	0.04	0.99	95,66%
IBK 3	0.96	0.05	0.99	94,93%
IBK 5	0.95	0.06	0.99	94,30%
SimpleLogistic	0.97	0.06	0.99	93,91%
NaiveBayes	0.93	0.14	0.96	86,81%
BayesNet K2	0.96	0.16	0.97	85,83%
BayesNet TAN	0.97	0.10	0.98	90,55%
SMO Poly	0.97	0.07	0.95	93,45%
SMO NPoly	0.97	0.05	0.96	94,70%
J48	0.96	0.05	0.98	95,06%
RandomTree	0.95	0.05	0.96	94,89%
RandomForest 10	0.97	0.04	0.99	95,63%
RandomForest 50	0.97	0.04	0.99	96,00%
RandomForest 100	0.97	0.04	0.99	96,05%

the viability of the usage of permissions as a mechanism to detect malware using machine-learning.

In order to validate the previous scope, we collected a total amount of 5,511 goodware samples. Also, we used the Drebin dataset as malware dataset. Using different classifiers, we generated the models and evaluated their configuration with the Area Under ROC Curve (AUC). We obtained a 0.99 of AUC using the Random Forest classifier.

In light of these results and in spite of the fact that in the future this approach could change, the viability of Sanz's approach has been demonstrated. Nowadays, Android uses a group of permissions, so, with this approach we do not know if a

malicious application is using all the permissions that it requests or not. However, this could be used as a first step before a more extensive analysis.

In future lines, as Drebin dataset is composed by families of malware, using their permissions and machine learning techniques, we could obtain their family. Besides, as the malware in Android is a problem that is growing exponentially every day,we believe that the creation of a dynamic tool by the community would be essential.