PUMA: Permission Usage to detect Malware in Android

Borja Sanz[1], Igor Santos[1], Carlos Laorden[1], Xabier Ugarte-Pedrero[1], Pablo Garcia Bringas[1], and Gonzalo Álvarez[2]

[1]S^{3}Lab, University of Deusto
Avenida de las Universidades 24, 48007 Bilbao, Spain
{borja.sanz, isantos, claorden, xabier.ugarte, pablo.garcia.bringas}@deusto.es

[2]Instituto de Física Aplicada, Consejo Superior de Investigaciones Científicas (CSIC)
Madrid, Spain
gonzalo@iec.csic.es

Abstract. The presence of mobile devices has increased in our lives offering almost the same functionality as a personal computer. Android devices have appeared lately and, since then, the number of applications available for this operating system has increased exponentially. Google already has its Android Market where applications are offered and, as happens with every popular media, is prone to misuse. In fact, malware writers insert malicious applications into this market, but also among other alternative markets. Therefore, in this paper, we present PUMA, a new method for detecting malicious Android applications through machine-learning techniques by analysing the extracted permissions from the application itself.

Keywords: malware detection, machine learning, Android, mobile malware

1 Introduction

Smartphones are becoming increasingly popular. Nowadays, these small computers accompany us everywhere, allowing us to check the email, to browse the Internet or to play games with our friends. It is necessary a need to install applications on your smartphone in order to take advantage of all the possibilities that these devices offer.

In the last decade, users of these devices have experienced problems when installing mobile applications. There was not a centralized place where users could obtain applications, and they had to browse the Internet searching for them. When they found the application they wanted to install, the 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 despite having bought them).

Nowadays there are new methods to distribute 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 was the first store to implement this new model and was very successful, but other manufacturers such as Google, RIM and Microsoft have followed the same business model developing application stores accessible from the device. Users only need now an account for an application store in order to buy and install new applications.

These factors have drawn developers' attention to these platforms. According to Apple[1], the number of available applications on the App Store is over 350,000, whilst Android Market[2] has over 200,000 applications.

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.

The 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 reviewers. In contrast, Android relies on its security permission system and on the user's sound judgement. Unfortunately, users have usually no security consciousness and they do not read required permissions before installing an application.

Although both AppStore and Android Market include clauses in the terms of services that urge developers not to submit malicious software, both have hosted malware in their stores. To solve this problem, they have developed tools for removing remotely these malicious applications. Both models are insufficient to ensure user's safety and new models should have been included in order to improve the security of the devices.

Machine learning techniques have been widely applied for classifying applications which are mainly focused on generic malware detection [1–5]. Besides, several approaches [6, 7] have been proposed to classify applications specifying the malware class; e.g., trojan, worms, virus; and, even the malware family.

With regards to Android, the number of malware samples is increasing exponentially and several approaches have been proposed to detect them. Shabtai et al. [8] trained machine learning models using as features the count of elements, attributes or namespaces of the parsed Android Package File (*.apk*). To evaluate their models, they selected features using three selection methods: Information Gain, Fisher Score and Chi-Square. They obtained 89% of accuracy classifying applications into only 2 categories: tools or games.

There are other researches that use a dynamic analysis to detect malicious applications. Crowdroid [9] is an earlier approach that analyse the behaviour

[1] `http://www.apple.com/iphone/features/app-store.html`

[2] http://googleblog.blogspot.com/2011/05/android-momentum-mobile-and-more-at.html

[3] `http://online.wsj.com/public/resources/documents/wsj-2009-0731-FCCApple.pdf`

of the applications. Blasing et al. [10] created AASandbox, which is an hybrid approximation. Dynamic part is based on the analysis of the logs for the low-level interactions obtained during execution. Shabtai et al. [11] also proposed a Host-Based Intrusion Detection System (HIDS) which uses a machine learning methods that determines if the application is malware or not.

On the other hand, Google has deployed a supervision framework, called "Bouncer", which analyses the applications before being published. Oberheide and Miller [12] has revealed some features of this system. For example, the systems is based in QEMU and make both static and dynamic analysis.

Given this background, we present PUMA, a new method for detecting malicious Android applications employing the permission usage of the each application. Using these features, we train a machine-learning models to detect whether an applications is malware or not. Summarising, our main findings in this paper are: (i) we describe the process of extracting features from the Android .apk files, (ii) we propose a new representation for Android applications in order to develop a malware detection approach, and (iii) we perform an empirical validation of our approach and show that it can achieve high accuracy rates. The reminder of this paper is organised as follows. Section 2 details the generation of the dataset. Section 3 presents the permissions used in our approach. Section 4 describes the empirical evaluation of our method. Finally, section 5 discusses the results and shows the avenues of further work.

2 Description of the dataset

2.1 Benign software

To conform this dataset, we gathered a collection of 1811 Android applications of different types. In order to classify them properly, we chose to follow the same naming as the official Android market. To this end, we used an unofficial Android Market API[4] to connect with the Android market and, therefore, obtain the classification of the applications.

We selected the number of applications within each category according to their proportions in the Android Market. There are several application types in Android: native applications (developed with the Android SDK), web applications (developed mostly with HTML, JavaScript ad CSS) and widgets (simple applications for the Android desktop, which are developed in a similar way to web applications). To generate the dataset, we did not make distinctions between these types and every of them is represented in the final dataset. Once we determined the number of samples for each category, we randomly selected the applications. The number of samples obtained for each category is shown in Table 1.

[4] `http://code.google.com/p/android-market-api/`

Table 1. Number of benign software applications.

Category	*Number*	*Category*	*Number*
Action and Arcade	33	Libraries and Demos	2
Races	3	Books and References	9
Casual	11	Medicine	2
Comics	1	Multimedia and Video	25
Shopping	3	Music and Audio	13
Communications	21	Business	2
Sports	4	News and Magazines	7
Education	1	Personalization	6
Companies	5	Productivity	30
Entertainment	16	Puzzles	16
Way of life	5	Health and Fitness	3
Accounting	2	Society	28
Photography	6	Weather	2
Tools	86	Transportation	2
Casino and card games	4	Sports and guides	9
	Total: 357		

2.2 Malicious software

Malware samples were gathered by means of *VirusTotal*[5] which is an analysis tool for suspect files, developed by *Hispasec Sistemas*[6], a company devoted to security and information technologies. We have used their service called *VirusTotal Malware Intelligence Services*, available for researchers to perform queries to their database.

Using this tool, we gathered a total number of 4,301 samples. However, we performed a duplication removal step, where we deleted from the dataset the duplicates samples. Finally, we used a total number of 249, which according to Lookout[7] represents the 54% of the total malware samples.

3 Permissions of Android Applications

We performed an study of the different permissions of the applications in order to determine their suitability for malware detection. These permissions are evaluated when installing the app, and must be approved by the user. We used the *Android Asset Packaging Tool* (aapt) to extract and decrypt the data from the `AndroidManifest.xml` file, provided by the Android SDK. Thereafter, we dumped the result and processed it. From all the information available, we only

[5] `http://www.virustotal.com`
[6] `http://www.hispasec.com/`
[7] `https://www.mylookout.com/_downloads/lookout-mobile-threat-report-2011.pdf`

used the following features: (i) "uses-permission", every permission that the application needs to work is defined under this tag; and (ii) "uses-feature", which shows which are the features of the device the application uses.

We have removed the rest of the information that the AndroidManifest file (shown in Figure 1) stores because of its dependency on specific devices.

```
<manifest>
      <uses-permission />
      <permission />
      <permission-tree />
      <uses-sdk />
       ....
</manifest>
```

Fig. 1. Example of AndroidManifest file.

Fig. 2 suggests that the most frequent permissions in both categories are the same. Besides, it seems that there are not visible differences in the permissions used in malware with respect to the ones used in benign applications, at least, when we studyied separately. In other words, malicious applications do not need different permissions than benign ones. This may indicate that the granularity of the permissions system is not accurate enough to distinguish malicious intentions.

On the other hand, we conducted a study regarding the number of permissions of each application (shown in Fig. 3). The number of permissions required for both malicious and benign applications is also nearly the same. However, we noticed several differences in both classes: the chance of finding malware applications requiring only one permission is high while benign applications usually present 2 or 3 permissions. This fact suggests that only one permission is needed to behave maliciously on Android device.

4 Empirical Validation

To validate PUMA, 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)[8]. In particular, we have used the classifiers specified in Table 2. To evaluate the performance of machine-learning classifiers, *k-fold cross validation* is usually used [13]. Thereby, for each classifier we tested, we performed a k-fold cross validation [14] 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).

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

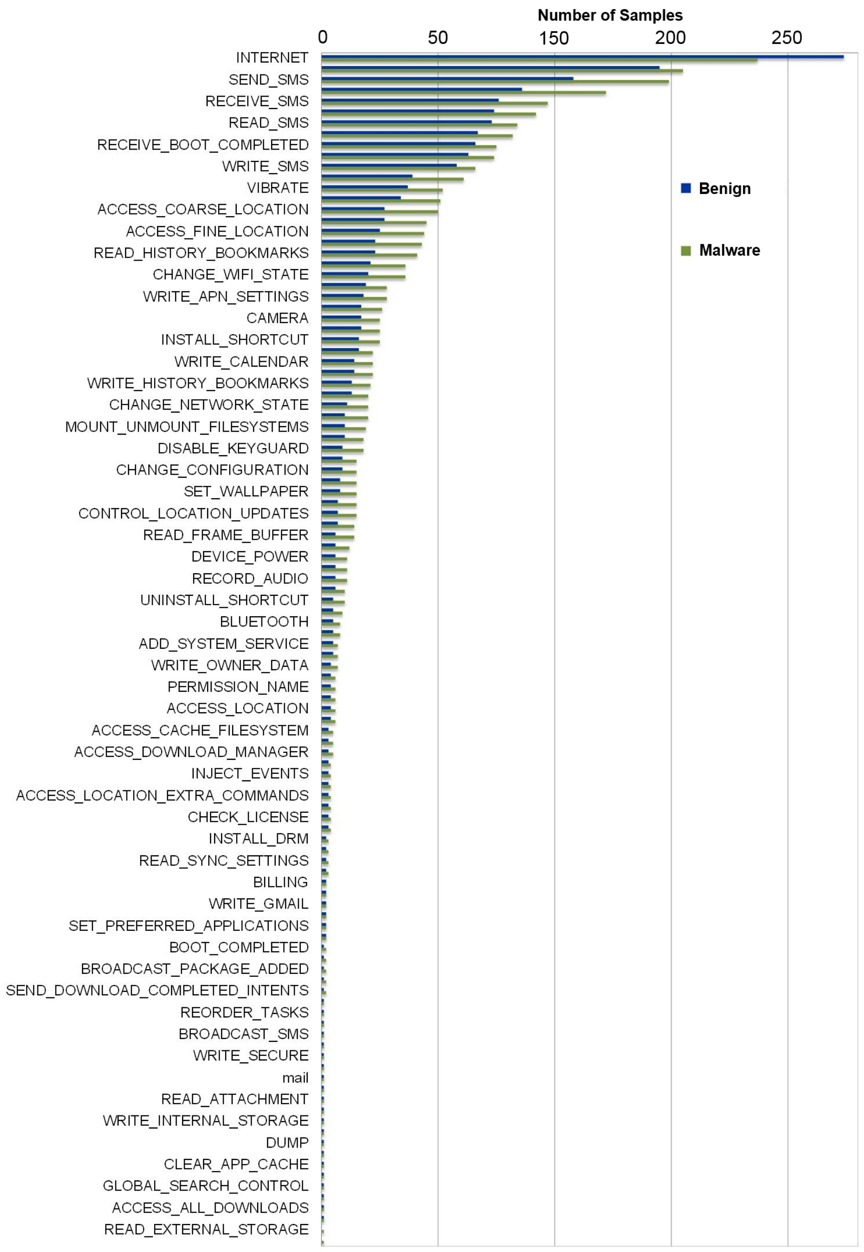

Fig. 2. Extracted permissions for the applications conforming the dataset.

To evaluate each classifier's capability, we measured the True Positive Ratio (TPR):

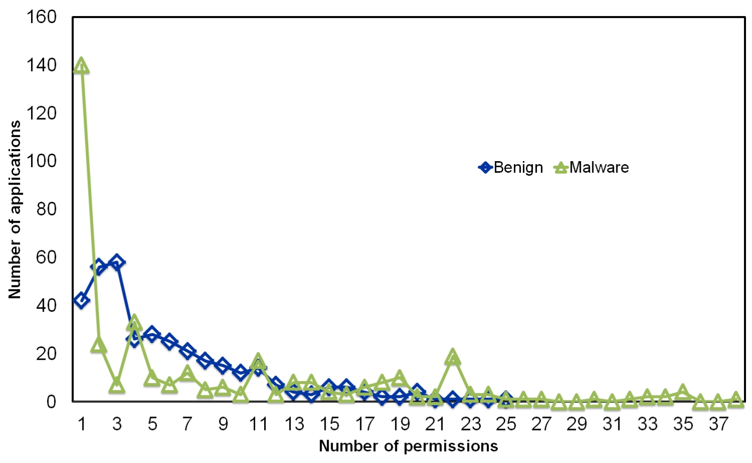

Fig. 3. Number of permissions of benign and malware apps.

Table 2. Machine learning classifiers to used in the experiment.

Algorithm	Used configuration
SimpleLogistic	
NaiveBayes	
BayesNet	K2 and TAN
SMO	PolyKernel y NormalizedPolyKernel
IBK	Valores de K: 1, 3 and 5
J48	
RandomTree	
RandomForest	Valor de I: 10, 50 and 100

$$TPR = \frac{TP}{TP + FN} \tag{1}$$

where TP is the number of malware cases correctly classified (true positives) and FN is the number of malware cases misclassified as legitimate software (false negatives).

We also measured the False Positive Ratio (FPR):

$$FPR = \frac{FP}{FP + TN} \tag{2}$$

where FP is the number of benign software cases incorrectly detected as malware and TN is the number of legitimate executables correctly classified.

Furthermore, we measured the accuracy, i.e., the total number of the classifier's hits divided by the number of instances in the whole dataset:

$$Accuracy = (TP + TN) \cdot \frac{TP + TN}{TP + FP + TP + TN} \tag{3}$$

Besides, we measured the Area Under the ROC Curve (AUC) which establishes the relation between false negatives and false positives [15]. The ROC curve is obtained by plotting the TPR against the FPR.

Table 3. Android malware detection results for the different classifiers.

Algorithm	TPR	FPR	AUC	Accuracy
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%
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%
IBK 10	0.85	0.27	0.87	78.94%
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%

Table 3 shows the obtained results. With the exception of the Bayesian-based classifiers, the methods achieved accuracy rates higher than 80%. In particular, the best classifier, in terms of accuracy, was Random Forest trained with 50 trees with a 86.41%. Regarding the TPR results, Random Forest trained with 10 trees was the best classifier with a 0.92. The lowest FPR was obtained with Bayesian networks trained with Tree Augmented Naïve, however, its TPR results are lower than 55%. In terms of AUC, Random Forest was the best classifier with a 0.92.

5 Discussion and Conclusions

Permissions are the most recognisable security feature in Android. User must accept them in order to install the application. In this paper we evaluate the capacity of permissions to detect malware using machine-learning techniques.

In order to validate our method, we collected 239 malware samples of Android applications. Then, we extracted the aforementioned features for each application and trained the models, evaluating each configuration using the Area Under ROC Curve (AUC). We obtained a 0.92 of AUC using the Random Forest classifier.

Nevertheless, there are several considerations regarding the viability of our approach. Forensic experts are developing reverse engineering tools over Android applications, from which researchers could retrieve new features to enhance the data used to train the models. Furthermore, despite the high detection rate, the obtained result has an high false positive rate. Consequently, this method can be used as a first step before other more extensive analysis, such as a dynamic analysis.

Future work of this Android malware detection tool is oriented in two main directions. First, there are other features from the applications that could be used to improve the detection ratio that do not require to execute the sample. Forensics tools for Android applications should be developed in order to obtain new features. Second, dynamic analysis provides additional information that could improve malware detection systems. Unfortunately, smartphones resources are limited and these kind of analysis usually consumes resources that these devices don't have.

References

1. Schultz, M., Eskin, E., Zadok, F., Stolfo, S.: Data mining methods for detection of new malicious executables. In: Proceedings of the 2001 IEEE Symposium on Security and Privacy. (2001) 38–49
2. Devesa, J., Santos, I., Cantero, X., Penya, Y.K., Bringas, P.G.: Automatic Behaviour-based Analysis and Classification System for Malware Detection. In: Proceedings of the 12th International Conference on Enterprise Information Systems (ICEIS). (2010) 395–399.
3. Santos, I., Nieves, J., Bringas, P.G.: Semi-supervised learning for unknown malware detection. In: Proceedings of the 4th International Symposium on Distributed Computing and Artificial Intelligence (DCAI). 9th International Conference on Practical Applications of Agents and Multi-Agent Systems (PAAMS). (2011) 415–422
4. Santos, I., Laorden, C., Bringas, P.G.: Collective classification for unknown malware detection. In: Proceedings of the 6th International Conference on Security and Cryptography (SECRYPT). (2011) 251–256
5. Santos, I., Brezo, F., Ugarte-Pedrero, X., Bringas, P.G.: Opcode sequences as representation of executables for data-mining-based unknown malware detection. Information Sciences **?**(?) ?–? doi:10.1016/j.ins.2011.08.020, in press.
6. Rieck, K., Holz, T., Willems, C., Düssel, P., Laskov, P.: Learning and classification of malware behavior. In: Proceedings of the 2008 Conference on Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA), Springer (2008) 108–125
7. Tian, R., Batten, L., Islam, R., Versteeg, S.: An automated classification system based on the strings of trojan and virus families. In: Malicious and Unwanted Software (MALWARE), 2009 4th International Conference on, IEEE (2009) 23–30
8. Shabtai, A., Fledel, Y., Elovici, Y.: Automated Static Code Analysis for Classifying Android Applications Using Machine Learning. 2010 International Conference on Computational Intelligence and Security (December 2010) 329–333
9. Burguera, I., Zurutuza, U., Nadjm-Tehrani, S.: Crowdroid: behavior-based malware detection system for android. In: Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices, ACM (2011) 15–26

10. Blasing, T., Batyuk, L., Schmidt, A., Camtepe, S., Albayrak, S.: An android application sandbox system for suspicious software detection. In: Malicious and Unwanted Software (MALWARE), 2010 5th International Conference on, IEEE (2010) 55–62
11. Shabtai, A., Elovici, Y.: Applying behavioral detection on android-based devices. Mobile Wireless Middleware, Operating Systems, and Applications (2010) 235–249
12. Oberheide, J., Miller, J.: Dissecting the android bouncer (2012)
13. Bishop, C.: Pattern recognition and machine learning. Springer New York. (2006)
14. Kohavi, R.: A study of cross-validation and bootstrap for accuracy estimation and model selection. In: International Joint Conference on Artificial Intelligence. Volume 14. (1995) 1137–1145
15. Singh, Y., Kaur, A., Malhotra, R.: Comparative analysis of regression and machine learning methods for predicting fault proneness models. International Journal of Computer Applications in Technology **35**(2) (2009) 183–193