Permission-Based Android Malware Detection

Zarni Aung, Win Zaw

Abstract: - Mobile devices have become popular in our lives since they offer almost the same functionality as personal computers. Among them, Android-based mobile devices had appeared lately and, they were now an ideal target for attackers. Android-based smartphone users can get free applications from Android Application Market. But, these applications were not certified by legitimate organizations and they may contain malware applications that can steal privacy information for users. In this paper, a framework that can detect android malware applications is proposed to help organizing Android Market. The proposed framework intends to develop a machine learning-based malware detection system on Android to detect malware applications and to enhance security and privacy of smartphone users. This system monitors various permissionbased features and events obtained from the android applications, and analyses these features by using machine learning classifiers to classify whether the application is goodware or malware.

Index Terms: - Smartphones , Android, Malware detection , Machine Learning,

1 INTRODUCTION

N the last years, mobile devices, such as smartphones, tablets and PDAS, have become popular by increasing the number and complexity of their capabilities. Current mobile devices offer a large amount of services and applications than those offered by personal computers. At the same time, the increasing number of security threats that target mobile devices has emerged. In fact, malicious users and hackers are taking advantage of both the limited capabilities of mobile devices and the lack of standard security mechanisms to design mobile-specific malware that access sensitive data, steal the user's phone credit, or deny access to some device functionalities. In 2011, malware attacks increased by 155 percent across all platforms [1]: in particular, Android is the platform with the highest malware growth rate by the end of 2011. To mitigate these security threats, various mobilespecific Intrusion Detection Systems (IDSes) have been recently proposed. Most of these IDSes are behavior-based, i.e. they don't rely on a database of malicious code patterns, as in the case of signature-based IDSes. In this paper, we describe a machine learning based malware detection system for android based smartphones users. This system exploits machine learning techniques to distinguish between normal and malware applications.

Summarising, our main findings in this paper are:

1. We describe the process of extracting features from the Android .apk files
2. We create a dataset from extracted features of Android applications in order to develop android malware detection framework
3. We perform an empirical validation of machine learning approaches and show that they can achieve high accuracy rates.

- Zarni Aung is currently pursuing Ph.D degree program in Information Technology in University of Technology (Yatanarpon Cyber City),Myanmar, Email: zarniaung.utycc@gmail.com
- Win Zaw is currently working as Head of Department of Information Technology in Technological University (Thanlynn), Myanmar, Email: winzaw@gmail.com

The remainder of the paper is organized as follows. Section 2 lists some related work. Section 3 discusses the malware detection techniques and Section 4 describes the implementation of overview system design. Section 5 experiments malware detection architecture step by step and concludes the system and proposes future work in Section 6.

2 RELATED WORK

Crowdroid[2] is a machine learning-based framework that recognizes Trojan-like malware on Android smartphones, by analyzing the number of times each system call has been issued by an application during the execution of an action that requires user interaction. A genuine application differs from its trojanized version, since it issues different types and a different number of system calls. Crowdroid builds a vector of m features (the Android system calls). Another IDS that relies on machine learning techniques is Andromaly [3] which monitors both the smartphone and user's behaviors by observing several parameters, spanning from sensors activities to CPU usage. 88 features are used to describe these behaviors; the features are then pre-processed by feature selection algorithms. The authors developed four malicious applications to evaluate the ability to detect anomalies. MADAM: a Multi-Level Anomaly Detector for Android Malware [5] uses 13 features to detect android malware for both kernal level and user level. MADAM has been tested on real malware found in the wild and uses a global-monitoring approach that is able to detect malware contained in unknown applications, i.e. not previously classified. [7] monitors smartphones to extract features that can be used in a machine learning algorithm to detect anomalies. The framework includes a monitoring client, a Remote Anomaly Detection System (RADS) and a visualization component. RADS is a web service that receives, from the monitoring client, the monitored features and exploits this information, stored in a database, to implement a machine learning algorithm.[8] proposes a behavior-based malware detection system (pBMDS) that correlates user's inputs with system calls to detect anomalous activities related to SMS/MMS sending. [9] and [10] propose Kirin security service for Android, which performs lightweight certification of applications to mitigate malware at install time. Kirin certification uses security rules that match undesirable properties in security configuration bundled with applications. [11] performs static analysis on the executables to extract functions calls usage using

readelf command. Hence, these calls are compared with malware executables for classification. Finally, [12] surveys some security solutions for mobile devices.

3 MALWARE DETECTION APPROACHES

There are three different types of malware detection techniques: attack or invasion detection, misuse detection (signature-based) and anomaly detection (behavior-based) [6]. Attack or Invasion detection tries to detect unauthorized access by outsiders. But, misuse detection (signature-based) tries to detect misuse by insiders and describes very good detection results for specified, well-known attacks. The advantages of misuse detection are: it has no false positives and can quickly detect intrusion. Disadvantage is not capable of detecting new unfamiliar intrusions, even if they are built as minimum variants of already known attacks. Anomaly detection (behavior-based) refers to detecting patterns in a given dataset that do not conform to an established normal behavior. It also attempts to estimate the abnormal behavior of the system to be protected and generate anomaly alarm whenever the deviation between a given observation at an instance and normal behavior exceeds a predefined threshold. Advantage is potential to detect previously unseen intrusion events and disadvantage is many false positives and requires a large set of training data to construct normal behavior profile. For removing these shortcomings of misuse detection and anomaly detection profiles should be updated with large amount the datasets at regular interval of time [16].But a large amount of the datasets also increases the problem of inconsistency, redundancy and ambiguity. Several data mining techniques have been applied for intrusion detection. K-Mean Clustering is an unsupervised data mining technique for intrusion detection and it is easy to implement. Three major drawback of K-mean clustering are: 1. class dominance problem, 2.force assignment problem, and 3. no class Problem. It has been observed that single model cannot give better result in terms of recall and precision.

4 IMPLEMENTATION

For removing all these mentioned problems are proposing a new model which is based on feature selection as a first phase, K-Mean clustering model generation as a second phase, classification of this new dataset which is generated by second phase as third phase, and finally evaluating the performance of this proposed model in terms of accuracy, precision and recall. The overview design of proposed architecture is shown in Figure 4.

4.1 Features

For each Android application, we retrieved several selected features from the corresponding application package (APK) file. In addition, we identified real permissions required by he application, and adopted the features for malware detection. The values of selected features are stored as a binary number (0 or 1), which is represented as a sequence of comma separated values. We enumerate all selected features in the following items. Each item includes the name of a feature, the data type of the feature, and data of the feature. The few sample features are described in the following:

1. android.permission.INTERNET-the application request INTERNET permission when it is installed. The user can allow this permission request because he/she does not know this permission request is important. Every application does not require this permission request. If an application does not require INTERNET permission but it request the permission, this application can not be defined as normal application. INTERNET permission request is one of the dangerous features because the malware application can send user privacy information to their websites.

2. android.permission.CHANGE_CONFIGURATION- the application can request to change configuration files of the mobile devices. The users can be blocked from the available services and functionalities of their mobile devices. The application can change the stored data files so that the storage data can be lost by allowing this permission request.

3. android.permission.WRITE_SMS- the application can write SMS message without user notification by allowing this permission request.

4. android.permission.SEND_SMS- the application can send SMS message so that the money can be lost by installing similar applications with this permission request.

5. android.permission.CALL_PHONE- some applications can request CALL_PHONE permission without necessary for them. If the user allows this permission request, the application will call phone itself without user notification. The users do not know why their money was lost without their usage. The other permission features are not described in this paper.

0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,5.673513,goodware

Figure1. An example feature vector for the good ware application retrieved from its Android application package file

0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,1,0,0,0,1,0,0,0,0,0,0,0,1,0
,0,0,0,0,1,0,0,0,1,0,0,0,1,1,0,0,0,1,0,0,0,0,0,1,0,0,1,1,0,0,0,
0,0,0,1,0,1,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,1,0,0
,0,0,0,1,0,1,0,0,0,0,1,0,0,1,0,0,0,0,0,0,0,1,0,1,0,0,0,0,0,0,1,
6.156783,malware

Figure2. An example feature vector for malware application retrieved from its Android application package file

4.2 Feature Extraction

In this section, we describe the process we followed to obtain data from the android application file. The general steps we have followed for each application are:

1. We downloaded and collected malware and good ware applications from application market.

2. We decompress applications to extract the content.

3. We extract the permission request features from each application.

4. We build a dataset in an ARFF [4] file format with the extracted data.

First, we decompress the android application package file to extract the content. During the first three steps we retrieve the information from this source. We process the AndroidManifest.xml file to extract these data.

4.3 Feature Selection

In Machine Learning applications, a large number of extracted features, some of which redundant or irrelevant, present several problems such as—misleading the learning algorithm, over-fitting, reducing generality, and increasing model complexity and run-time. These adverse effects are even more crucial when applying Machine Learning methods on mobile devices, since they are often restricted by processing and storage-capabilities, as well as battery power. Applying fine feature selection in a preparatory stage enabled to use our malware detector more efficiently, with a faster detection cycle. Nevertheless, reducing the amount of features should be performed while preserving a high level of accuracy. In this section we select the k best features from the extracted features of android application package files by using feature selection method: Information Gain. This method depends on entropy of the attributes and it selects the largest value of gain as the best feature. Gain of an attribute A on a collection of examples S is given by

$$Gain(S, A) = Entropy(S) - \sum_{V \in Values(A)} \frac{|SV|}{|S|} Entropy(SV)$$

4.4 Machine Learning and Malware Detection

The selected features are collected into the signature database and divided into training data and test data and used by standard machine learning techniques to detect the android malware applications. We choose K-means clustering (i) it is data driven method relatively few assumptions on the distributions of the underlying data and (ii) it guarantees at least a local minimum of the criterion function, thereby accelerating the convergence of clusters on large datasets. First stage: clustering is performed on training instances to obtain k disjoint clusters. Each cluster represents a region of similar instances in terms of Euclidean distances between the instances and their cluster centroids. Second stage: K-means method is cascaded with decision tree learning by using the instances in each K-means cluster. Disadvantage of K-means:

1. very sensitive to noise, mixed pixels and outliers

2. limit the application to only numerical variables

3. anomaly type of data is not eliminated if the overlapping is found

If the overlapping is found in k-means cluster, decision tree classifiers are used to classify each cluster

K-Means Algorithm

1. Select k centroids arbitrarily (called as seed) for each cluster Ci , i ε [1, k]

2. Assign each data point to the cluster whose centroid is closest to the data point.

3. Calculate the centroid Ci of cluster Ci, i ε [1, k].

4. Repeat steps 2 and 3 until no points change between clusters.

1. Select k random instances from the training data subset as the centroids of the clusters C1; C2; ...Ck.

2. For each training instance X:
   a. Compute the Euclidean distance D(Ci,X), i=1... k:Find cluster Cq that is closest to X.
   b. Assign X to Cq. Update the centroid of Cq.(The centroid of a cluster is the arithmetic mean of the instances in the cluster.)

3. Repeat Step 2 until the centroids of clusters C1;C2; ...Ck stabilize in terms of mean-squared- error criterion.

4. For each test instance Z:
   a. Compute the Euclidean distance D(Ci,Z), i=1... k. Find cluster Cr that is closest to Z.
   b. Classify Z as an anomaly or a normal instance using the Threshold rule

The Threshold rule for classifying a test instance Z that belongs to cluster Cr is:
Assign Z-->1 if P(w|Z)> Threshold;
Otherwise Z →0 where “0” and “1” represent normal and malware classes [6]

Figure 3. K-Means Clustering Algorithm

Decision tree technology is a common, intuitionist and fast classification method [15]. Its construction process is top-down, divide-and rule. Essentially it is a greedy algorithm. Starting from root node, for each non-leaf node, firstly choose an attribute to test the sample set; Secondly divide training sample set into several sub-sample sets according to testing results, each sub-sample set constitutes a new leaf node; Thirdly repeat the above division process, until having reached specific end conditions. In the process of constructing decision tree, selecting testing attribute and how to divide sample set are very crucial. Different decision tree algorithm uses different technology. In practice, because the size of training sample set is usually large, the branches and layers of generated tree are also more. In addition, abnormity and noise existed in training sample set will also cause some abnormal branches, so we need to prune decision tree. One of the greatest advantages of decision tree classification algorithm is that: It does not require users to know a lot of background knowledge in the learning process.

J48 Decision Tree Algorithm
Generate a decision tree from the given training data

Input: training sample set T, the collection of candidate attribute attribute_list

Output: a decision tree.

1. Create a root node N;
2. If T belongs to the same category C, then return N as a leaf node, and mark it as class C;
3. If attribute_list is empty or the remainder samples of T is less than a given value, then return N as a leaf node, and mark it as the category which appears most frequently in attribute_list, for each attribute, calculate its information gain ratio
4. Suppose test_attribute is the testing attribute of N, then test_attribute= the attribute which has the highest information gain ratio in attribute list:
5. If testing attribute is continuous, then find its division threshold;
6. For each new leaf node grown by node N

   {

   a. Suppose T is the sample subset corresponding to the leaf node.

   b. If T has only a decision category, then mark the leaf node as this category;

   c. Else continue to implement J48_Tree (T',T'_attributelist)

   }
7. Calculate the classification error rate of each node and then prune the tree.

Random forests (RF) are a combination of tree predictors such that each tree depends on the values of a random vector sampled independently and with the same distribution for all trees in the forest. The generalization error of a forest of tree classifiers depends on the strength of the individual trees in the forest and the correalation between them. Using a random selection of features to split each node yields error rates that compare favorably to Adaboost, and are more robust with respect to noise.

Random forest Algorithm (A variant of bagging)

1. Select ntree, the number of trees to grow, and mtry, a number no larger than number of variables.
2. For i = 1 to ntree:
3. Draw a bootstrap sample from the data. Call those not in the bootstrap sample the "out-of-bag" data.
4. Grow a "random" tree, where at each node, the best split is chosen among mtry randomly selected variables. The tree is grown to maximum size and not pruned back.
5. Use the tree to predict out-of-bag data.
6. In the end, use the predictions on out-of-bag data to form majority votes.
7. Prediction of test data is done by majority votes from predictions from the ensemble of trees.

Classification and Regression Tree (CART)

Splitting rule:
Choose the split that maximizes the decrease in impurity.
Impurity:

1. Gini Index

$$\phi(p) = \sum_{k \neq l} p_k p_l = 1 - \sum_{k \neq l} p_k^2$$

2. Entropy

$$\phi(p) = -\sum_{k} p_k \log p_k$$

Split stopping rule:
A large tree is grown and procedures are implemented to prune the tree up-ward.

Class assignment:
Normally simply assign the majority class in the node unless a strong prior of the class probability is available.

5 EXPERIMENTS

We tested our system against a collection of 500 sample Android applications. We created two datasets from 200 and 500 android applications by extracting features and used Weka tool to analyse the evaluation of the proposed framework.

5.1 Performance Evaluation Criteria

We used the machine learning techniques to classify the malware applications. Firstly, we built ARFF file from the extracted features and train the dataset by using K-means clustering algorithm. Once the training model has been developed, we used decision tree learning algorithms for each cluster to classify the malware applications. From the response of classifiers, relevant confusion matrices were created. The following four metrics define the members of the matrix.

True Positive (TP): Number of correctly identified goodware applications.

False Positive (FP): Number of wrongly identified malware applications.

True Negative (TN): Number of correctly identified malware applications.

False Negative (FN): Number of wrongly identified goodware applications.

True Positive Rate (TPR): Percentage of correctly identified goodware applications

$$(TP / TP+FN)$$

False Positive Rate (FPR): Percentage of wrongly identified malware applications

$$(FP / TN+FP)$$

Overall Accuracy (ACC): Percentage of correctly identified applications

$$(TP+TN / TP+TN+FP+FN)$$

The performances of machine learning techniques were evaluated using the true positive rate, false positive rate and overall accuracy which are defined above.

5.2 Experimental Results

Firstly, we extracted the necessary features to analyze from sample applications (goodware and malware). Then, we built dataset in (.arff) file format from the extracted features. We used these two datasets to distinguish malware and goodware applications by machine learning approaches. Table 1 shows the details of two datasets used in android malware detection framework and the experimental results of different machine learning approaches from two datasets is shown in Table 2.

6 CONCLUSION

In this paper, we implement a framework for classifying Android applications using machine-learning techniques whether they are malware or normal applications. To generate the models, we have extracted several permission features from several downloaded applications from android markets. Some of the malware applications are used from malware sample database and both malware and normal applications are classified by using machine learning techniques. In order to validate our methods, we have collected 200 samples of Android applications and we have extracted the aforementioned features for each application and we have trained the models which have been evaluated using the Area Under ROC Curve (AUC). Regarding future work, we will train models with larger dataset as soon as we obtain enough samples of malicious applications and we will extract more features from sample applications. We will even classify the types of malware applications (Trojan, Infosteal, etc).

ACKNOWLEDGMENT

I would like to thank my supervisor, Dr. Win Zaw, Associate Professor and Head of Department of Information Technology, for his excellent guidance, where the initial scope of the thesis was defined, and throughout the process of writing this thesis.

REFERENCES

[1]. Juniper Networks: 2011 Mobile Threats Report(February 2012)

[2]. I. Burguera, U.Z., Nadijm-Tehrani, S.: Crowdroid: Behavior- Based Malware Detection System for Android. In: SPSM'11, ACM(October 2011)

[3]. A. Shabtai, U. Kanonov, Y. Elovici, C. Glezer, Y. Weiss: Andromaly: a behavioral malware detection framework for android devices. Journal of Intelligent Information Systems 38(1) (January 2011) 161-190

[4]. G. Holmes, A. Donkin, and I.H. Witten, "Weka: a machine learning workbench," August 1994, pp. 357-361.

[5]. G. Dini, F.Martinelli, A. Saracino, D. Sgandurra: MADAM: a Multi-Level Anomaly Detector for Android Malware

[6]. K. Hanumantha Rao, G. Srinivas, A. Damodhar, M. Vikas Krishna: Implementation of Anomaly Detection Technique Using Machine Learning Algorithms: Internatinal Journal of Computer Science and Telecommunications (Volume2, Issue3, June 2011)

[7]. Schmidt, A.D., Peters, F., Lamour, F., Scheel, C., Camtepe, s.A., Albayrak, S.: Monitoring smartphones for anomaly detection. Mob. Netw. Appl. 14(1)(2009) 92-106

[8]. Xie,L.,Zhang,X.,Seifert, J.P.,Zhu, S.: pBMDS: a behavior-based malware detection system for cellphone devices. In: Proceedings of the Third ACM Conference on Wireless Network Security, WISEC 2010, Hoboken, New Jersey, USA, March 22-24 2010, ACM(2010) 37-48

[9]. Enck, W., Ongtang, M., McDaniel, P.: On lightweight mobile phone application certification. In: CCS '09: Proceedings of the 16th ACM conference on Computer and Communication Security, New York, NY, USA, ACM (2009) 235-245

[10]. Ongtang, M., McLaughlin, S., Enck, W., McDaniel, P.: Semantically Rich Application-Centric Security in Android. In: Computer Security Applications Conference, 2009. ACSAC '09. Annual.(Dec 2009) 340-349

[11]. Schmidt, A.D., Bye, R., Schmidt, H.G., Clausen, J.H., Kiraz, O., Yuksel, K.A., Camtepe, S.A., Albayrak, S.: Static Analysis of Executables for Collaborative Malware Detection on Android. In: Proceedings of IEEE International Conference on Communications, ICC 2009, Dresden, Germany, 14-18 June 2009, IEEE (2009) 1-5

[12]. La Polla, M., Martinelli, F., Sgandurra, D.: A survey on security for mobile devices. Communications Surveys Tutorials, IEEE PP(99) (2012) 1-26

[13]. Freke, J.: smali - an assembler/disassembler for android's dex format. Google Project Hosting [online] http://code.google.com/p/smali/

[14]. Felt, A.P., Chin, E., Hanna, S., Song, D., Wagner, D.: Android permissions demystified. In: Proceedings of the 18th ACM conference on Computer and communications security. (2011) 627–638

[15]. Juan Wang, Qiren Yang, Dasen Ren, "An intrusion detection algorithm based on decision tree Technology", Asia-Pacific Conference on Information Processing, APCIP 2009, Shenzhen, IEEE 18-19 July 2009. pp. 333-335

[16]. Mark A. Hall, Lloyd A. Smith, "Feature Subset Selection: A Correlation Based Filter Approach 1997

Table 1. Datasets for Malware Detection Framework

Dataset Name	Number of Samples	Number of Features
Dataset #1	200	160
Dataset #2	500	160

Table 2. Experimental Results of Two Datasets

Dataset Name	Method Name	TP Rate	FP Rate	Precision	Recall	ROC Area	Correctly Classified Instances(%)	Incorrectly Classified Instances(%)
Dataset#1	J48	0.907	0.086	0.916	0.907	0.918	90.72%	9.28%
Dataset #1	Random Forest	0.918	0.081	0.918	0.918	0.954	91.75%	8.25%
Dataset#1	CART	0.978	0.157	0.849	0.978	0.87	90.72%	9.27%
Dataset #2	J48	0.88	0.121	0.88	0.88	0.915	88%	12%
Dataset #2	Random Forest	0.916	0.084	0.916	0.916	0.969	91.58%	8.42%
Dataset#2	CART	0.851	0.151	0.851	0.851	0.878	85.05%	14.94%

Figure 4. Android Malware Detection Framework