Malware Detection in Android Mobile Platform using Machine Learning Algorithms

Mariam Al Ali[1], Davor Svetinovic[2], Zeyar Aung[3], Suryani Lukman[4]

Khalifa University of Science and Technology
Abu Dhabi, United Arab Emirates
{maralali,dsvetinovic, zaung}@masdar.ac.ae, suryani.lukman@kustar.ac.ae

Abstract: Malware has always been a problem in regards to any technological advances in the software world. Thus, it is to be expected that smart phones and other mobile devices are facing the same issues. In this paper, a practical and effective anomaly based malware detection framework is proposed with an emphasis on Android mobile computing platform. A dataset consisting of both benign and malicious applications (apps) were installed on an Android device to analyze the behavioral patterns. We first generate the system metrics (feature vector) from each app by executing it in a controlled environment. Then, a variety of machine learning algorithms: Decision Tree, K Nearest Neighbor, Logistic Regression, Multilayer Perceptron Neural Network, Naive Bayes, Random Forest, and Support Vector Machine are used to classify the app as benign or malware. Each algorithm is assessed using various performance criteria to identify which ones are more suitable to detect malicious software. The results suggest that Random Forest and Support Vector Machine provide the best outcomes thus making them the most effective techniques for malware detection.

Keywords: malware detection, Android, apps, classification, machine learning

I. INTRODUCTION

Malware pose significant threats to cyber security of national infrastructures, service sectors, and ultimately the society as a whole. Nowadays, mobile devices like smart phones and tablets have become widely popular and have become sophisticated devices which can perform various tasks that a normal desktop computer could perform. Various organizations including government and bank organizations have implemented applications to be used in these devices. Personal and crucial information is now easily accessed through smart phones, in addition most of these information is stored in the cloud.

As a result, malware attacks have extended to mobile devices and it has become a necessity to protect ourselves from such attacks. Traditional signature-based and change-based malware detection methods are not able to cope with new types of malware attacks. In order to address this problem, in this research project, we plan to develop a practical and effective "anomaly-based" malware detection system with an emphasis on mobile computing platform. We carry out generation of system metrics (i.e., feature vector) and a variety of efficient machine learning techniques to fulfill our objective.

This paper is based on the M.Sc. thesis of the first author [1].

II. BACKGROUND

Early detection is the most important thing to mitigate the harmful effects of malware. Throughout the years, a number of malware detection methods has been proposed. These can be broadly categorized into: signature-based, change-based, and anomaly-based methods.

1) Signature-based methods: A signature based intrusion method detects a malware based on its signature. It first gathers data and analyzes it, and when a program or file has a similar signature to an already existing malware (which it compares to from a database) it detects it. This method is often used for detecting popular malware signatures, but it can be quite slow since it should compare the signatures from a large database, meaning it cannot be instant [24].

2) Change-based methods: Change based detection is a method that identifies when changes occurred in the system. It relies on probability distribution to detect the changes. These techniques include, online and offline change detection techniques.

3) Anomaly-based methods: "In the anomaly based system, a system administrator defines the baseline, or normal state of the network's traffic load, breakdown, protocol, and typical packet size. The anomaly detector monitors network segments to compare their state to the normal baseline and look for anomalies" [24].

Up until now, virtually almost all real-world deployments of malware detection (like virus scanners) are signature-based and change-based methods. Though efficient, these methods are not able to identify new types of malware (such as those carrying out zero-day attacks). Some research has been done on anomaly-based malware detection, which are good at

978-1-5386-0514-1/17/$31.00 ©2017 IEEE

identifying new malware. However, so far the anomaly-based methods are not widely deployed yet because of practical issues such as efficiency/scalability, high false positive rate, difficulty to use, etc. Our research objective is to fill in this research gap and propose a practical and effective anomaly-based malware detection method by comparing the different machine learning algorithms, especially in the new and emerging area of mobile computing.

III. RELATED WORKS

In order to detect malware in smart phones, several systems were proposed. As mentioned Andromaly [19] is such a system, in which a host-based malware detection system was created. This system monitored the phone and obtained information from it which was then analyzed using machine learning techniques. The information it obtained from the system was based on the behavior of the system thus making Andromaly a behavioral based detection system. The algorithms used include: k-means, LR, Histograms, Decision Trees, Bayesian Networks, and NB. Each of the results from classifying the data using these data mining algorithms were compared and it was determined that NB performed the best in some scenarios while decision trees, LR, and NB performed best in others. However, in their experiment Android was quite new so not many applications that contained malware were developed for it. Thus, the researchers had to develop two malware applications and install them to get the data.

Another important research conducted on behavioral detection is M0Droid [6], which consisted of a client and a server. The server analyzed the data collected from the client side to detect malware. This would reduce the battery consumption of having the client side analyze the data.

Crowdroid [5] also used anomaly based detection, however their methods were somewhat different. CrowDroid used two types of datasets, one they developed for testing and the other from real malware. The research concluded that detecting malware through monitoring system calls would work for emerging and new malicious software.

Aung and Zaw [2] proposed a model which first uses feature selection then applies k-means, next classifies the dataset using RF and J48 and finally analyzes it.

RiskRanker [10] used a method called zero-day detection, in which it checks through applications in the Android Market to determine which may be risky. This way the system narrows down the ”potential risks”. There are two order modules that RiskRanker follows, the “first-order module handles non-obfuscated apps by evaluating the risks” [10] while the second-order module analyzes the behavior of these applications in order to determine whether they are malicious or not. So, RiskRanker should be able to determine whether an app is dangerous upon installation thus it is called zero day detection.

ScanDroid [9], which examined the data flows of the Android application to determine if it is benign or not. One of the main aspects of these scans is checking the permissions of the app, as well as the certifications. Like RiskRanker, ScanDroid should detect malware as soon as it is installed on the system.

The study which developed DroidMat [23] detects malware through the manifest and API calling. First, the information is extracted from the manifest. Then the data is examined by checking the API calls made in the system. The analysis of the data is done by applying the k-means algorithm.

Other systems developed for malware detection in [20] used data mining algorithms such as SVM, NB, and Decision Trees. The researchers wanted to develop an android detection system which utilizes those algorithms for detection. The behavior of the Android phone is studied by collecting information from the APIs and libc functions, after through which it uses the data mining algorithms. The results of [20] showed that those algorithms were sufficient in detecting malware.

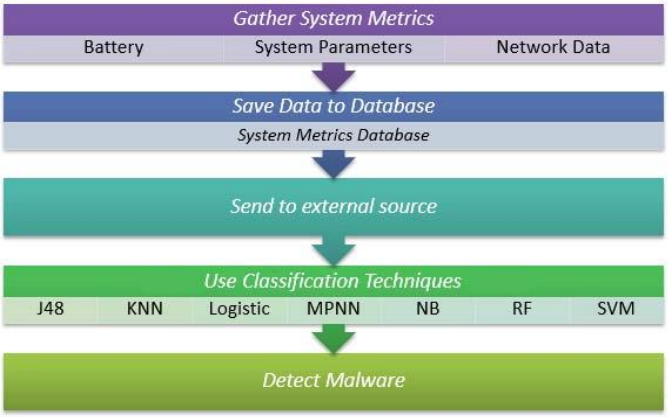

Fig. 1. Framework of proposed method

NB was used for anomaly based detection in a research conducted in [18]. In this study, an active platform runtime security system [18] was developed which checks the signatures of the applications installed to determine whether it is malware.

Other machine learning classifiers were studied to detect botnets in [8]. The classifiers studied included: NB, KNN, J48 decision tree, Multilayer Perceptron Neural Network, and SVM. The result of the experiment conducted showed that in regards to the most efficient classifier, KNN proved to have the best results. The results for KNN were somewhat surprising since the detection rate was 99%.

IV. PROPOSED METHOD

Figure 1 shows the stages in our proposed system model which would ultimately lead to the goal of detecting malware.

A. Dataset

The dataset used is the same one used in Damshenasa et al. on M0droid [6]. It contains a set of both benign and malicious software. The dataset comprises of 200 goodware and 115 malware apps, which will be installed on the mobile device. It compromises of a variety of ".apk" files such as games, widgets, and other apps. It should be noted that not all ".apk" files in the dataset will be used since some were not compatible with the operating system of the device that we use (unrooted Google/LG Nexus 4 running Android Lollipop version 5.1.1). As a result, the number of goodwares we used is 176 and that of malware is 59.

B. Generating System Metrics

The apps from the dataset are installed on the device and the data are collected. The program used to collect the data is developed in Android Studio. We allow the data collection to last approximately 60 seconds in order for the network data collection to be accurate.

Later on, we uninstall the app before installing a new one in order to avoid confusions in the results, i.e., we do not want the data from the other application to affect the other. Additionally, we allowed some time for the device to rest since we were worried about overheating from the battery and thus getting inaccurate results. From the data collected from each app, we derive the system metrics (i.e., the feature vector) that reflect the currentbehavior of the system. These include Battery Status, SystemParameters, and Network data as summarized in Table I.

1) **Battery Data:** They include the battery level, temperature, voltage, charger status, and battery status. Sudden changes in the battery implies that there is something wrong with the system, more probably a malicious software has infected it. Some studies showed that there are some malware applications that target battery life in mobile applications [14].

2) **System Parameters:** They consist of user parameters, the memory consumption and the total CPU consumption by the system and the app itself. Additionally, it contains the IOW and the IRQ which have to do with the memory management of the system. Malware is known to slow down the system thus effecting memory and CPU management, therefore checking the system parameters and showing that a major change has occurred could indicate that malware is present in the system.

3) **Network Data:** They include the data conveyed through Wi-Fi or mobile network services. The number of bytes that are currently transmitted or received is calculated. Most malware send personal data to external sources or let the device receive harmful data, so if a major increase occurs then it could indicate that the device is infected.

TABLE I: GENERATED SYSTEM METRICS

Battery Data	System Parameters	Network Data
Battery Level	System CPU	Received Packets
Temperature	System Memory	Transmitted Packets
Voltage	App CPU	
Charger Status	App Memory	
Battery Status	IOW	
	IRQ	

The generated system metrics are for each app is stored in a database called System Metrics Database (SMD). In SMD, the tables are divided according to the metrics mentioned. The records from SMD is later sent to an external source (like a central server) where the feature vector of the app's system metrics is analyzed. In the server, SMD records are analyzed using machine learning algorithms in order to classify the app as either malicious software or good software.

C. Classification Using Machine Learning Algorithms

A lightweight anomaly based detection system is proposed for Android mobile phones in this research paper. The idea is to take in the behavior of the system by analyzing various system metrics then evaluating them using machine learning algorithms in order to detect whether the system is infected with malware.

The framework proposed is similar to the one presented in Shabtai et al. [19], in which the system metrics were collected and then analyzed using classification algorithms for detection which included: k-means, Logistic Regression, Histograms, Decision Tree, Bayesian Networks and Naive Bayes. However, the key difference is that the classifiers in [19] were used within the device itself that proved to have its disadvantages such as consuming battery life [19]. In our work, we send the data to an external source (like a central server) and machine learning algorithm(s) for classification are executed in the server. Also the machine learning algorithms used in our work are different. These include: J48 (decision tree), KNN, Logistic, Naive Bayes, Multilayer Perceptron Neural Network, Random Forest, and Support Vector Machine.

We use Weka [11] to execute the machine learning algorithms. The brief description of each machine learning algorithm is given below.

1) **Decision Tree (J48):** The J48 algorithm [3] is a slightly modified C4.5 algorithm used in Weka. J48 is a decision tree which contains branches, a root, nodes, and leaves. Each node in the tree represents an attribute. The decision tree grows a tree using a depth-first strategy in order to generate a classification decision tree, to do this the algorithm splits the data in the tree, accordingly to get the best information gain. This algorithm is executed recursively until the tree ends.

2) **K Nearest Neighbor (KNN):** It is a lazy classifier. A lazy classifier means that the learner delays the work or the classification until a query is made, before that it tries to generalize the dataset [21]. The KNN classifier is a widely used classifier that is based on comparison methods, in which a test instance is compared with similar training instances, and the majority class of the nearest k instances is taken. In Weka, the KNN algorithm we used is called IBK, in which the value of k can be specified and different results are displayed accordingly. We try various *k*values: 1, 5, 10, 15, and 20. For our dataset, it turns out that k = 10 offers the best results.

3) **Logistic Regression (LR):** The LR model is generally used for the analysis of data with discrete variables which may have two or more possible variables. The results of a LR model are often binary [12].

4) **Multilayer Perceptron Neural Network (MPNN):** MPNNs are quite fast, easy to use, operate and implement, and require a small dataset thus making them fairly popular to use in data analysis [16]. The network is represented by nodes which both outputs and inputs vectors, additionally the nodes are connected together by weights. For training the data set, MPNNs uses back-propagation.

5) **Naive Bayes (NB):** The most common technique used for anomaly detection is Naive Bayes (NB), since several studies show that it is the most effective one [13]. The NB algorithm is the posterior probability in which a data instance t_i can be labeled c_j. In the equation 1 shown below, p is the number of attributes.

$$P(t_i|c_j) = \prod_{k=1}^{p} P(x_{ik}|c_j) \quad (1)$$

6) **Random Forest (RF):** An RF is a combination of several decision trees in which each tree in the forest is dependent on the value. It should be noted that the value is random vector samples independently and with the same distribution for all the trees in the forest [4]. The error rate of RF depends on the trees themselves, in which it can yield positive results in terms of error, strength, and correlation.

7) **Support Vector Machine (SVM):** SVM is a classifier that is based on the structural risk minimization (SRM) principle, which is achieved through a minimization of the upper bound of the generalization error [22]. SVM uses a hypothesis space of linear functions in a high dimensional feature space, in which it strives to achieve linear separability. In order to do that it applies margin maximization and kernels [15]. We try two versions of SVM in Weka, namely, Sequential Minimal Optimization (SMO) (with Puk kernel) and LibSVM (with linear kernel). For our dataset, it turns out that SMO offers the best results.

TABLE II: COMPARISON OF PERFORMANCES BY DIFFERENT MACHINE LEARNING METHODS. THE BEST RESULT FOR EACH CRITERIA IS HIGHLIGHTED IN BOLD

Performance Criteria	J48	KNN ($k = 10$)	LR	MPNN	NB	RF	SVM (SMO)
TP Rate	0.8831	0.8957	0.8809	0.8736	0.7531	**0.9060**	0.8933
FP Rate	0.2318	0.2400	0.2056	0.2249	**0.2000**	0.2425	0.2071
Precision	0.8782	0.8941	0.8798	0.8722	0.8198	**0.9063**	0.8910
Recall	0.8801	0.8957	0.8809	0.8736	0.7531	**0.9060**	0.8933
F-measure	0.8780	0.8918	0.8804	0.8726	0.7681	**0.9015**	0.8914
AUROC	0.8467	0.9184	0.9169	0.9123	0.8314	**0.9529**	0.9260

V. RESULTS AND DISCUSSIONS

On the dataset mentioned above in Section IV-A, we applied the various machine learning algorithms descried above in Section IV-C. We run 10-fold cross validation in our experiment. In order to ensure robustness of the results, for each machine learning algorithm, we conduct 10 sub-experiments using 10 different random seed values and take the average of the results.

The results in terms of six standard performance criteria [7], [17], namely, (i) True Positive (TP) Rate, (ii) False Positive (FP) Rate, (iii) Precision, (iv) Recall, (v) F-measure, and (vi) Area Under the Receiver Operating Characteristic Curve (AUROC), are presented in Table II.

It turns out that the Random Forest (RF) algorithm offers the best results in 5 out of 6 performance criteria (except for FP Rate). Figure 2 demonstrates a the random forest generated with a random seed of 2. SVM (SMO) is the the second best in terms of AUROC and is also quite decent in terms of other criteria. Its FP Rate is much better than that of RD. On the other hand, NB offers the best (lowest) False Positive Rate,

but its True Positive Rate and performance in other criteria are quite poor.

VI. CONCLUSIONS

The importance of the security of mobile devices like smart phones has increased throughout the years, specifically since more sensitive data is being stored in them. For that reason, this research proposed framework for malware detection in Android mobile systems. For each app, we collect the data on its system metrics and put them into a database called SMDB, and send it to a server. The database included the system metrics on battery data, system parameters and network data. The data is then analyzed in the server using machine learning algorithms which included Decision Tree, K Nearest Neighbor, Logistic Regression, Multilayer Perceptron Neural Network, Naive Bayes, Random Forrest, and Support Vector Machine. The results obtained and the comparisons made showed that Random Forest and Support Vector Machine (Sequential Minimal Optimization in particular) offer the best results among the algorithms examined.

REFERENCES

[1] M. Al Ali, "Malware detection in Android mobile platforms using data mining algorithms," Master's thesis, Masdar Institute of Science and Technology, Abu Dhabi, UAE, 2016.

[2] Z. Aung and W. Zaw, "Permission-based android malware detection," *International Journal of Scientific and Technology Research*, vol. 2, pp. 228–234, 2013.

[3] N. Bhargava, G. Sharma, R. Bhargava, and M. Mathuria, "Decision tree analysis on J48 algorithm for data mining," *International Journal of Advanced Research in Computer Science and Software Engineering*, vol. 3, pp. 1114–1119, 2013.

[4] L. Breiman, "Random forests," *Machine Learning*, vol. 45, pp. 5–32, 2001.

[5] I. Burguera, U. Zrutuza, and S. Nadjm-Tehrani, "Crowdroid: Behavior-based malware detection system for Android," in *Proceedings of the 1st ACM Workshop on Security and Privacy in Smartphones and Mobile Devices*, 2011, pp. 15–26.

[6] M. Damshenasa, A. Dehghantanha, K.-K. R. Choo, and R. Mahmud, "M0Droid: An Android behavioral-based malware detection model," *Journal of Information Privacy and Security*, vol. 11, pp. 141–157, 2015.

[7] T. Fawcett, "An introduction to ROC analysis," *Pattern Recognition Letters*, vol. 27, pp. 861–874, 2006.

[8] A. Feizollah, N. B. Anuar, R. Salleh, F. Amalina, R. R. Ma'arof, and S. Shamshirband, "A study of machine learning classifiers for anomaly-based mobile botnet detection," *Malaysian Journal of Computer Science*, vol. 26, pp. 251–265, 2014.

[9] A. P. Fuchs, A. Chaudhuri, and J. S. Foster, "Scandroid: Automated security certification of Android," Computer Science Department, University of Maryland, USA, Tech. Rep., 2009.

[10] M. Grace, Y. Zhou, Q. Zhang, S. Zou, and X. Jian, "Riskranker: Scalable and accurate zero-day Android malware detection," in *Proceedings of the 10th International Conference on Mobile Systems, Applications, and Services*, 2012, pp. 281–294.

[11] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The WEKA data mining software: An update," *SIGKDD Explorations*, vol. 11, pp. 10–18, 2009.

[12] D. W. Hosmer, S. Lemeshow, and S. X. Rodney, *Applied Logistic Regression*. John Wiley & Sons, 2013.

[13] M. Karim and R. M. Rahman, "Decision tree and naive bayes algorithm for classification and generation of actionable knowledge for direct marketing," *Journal of Software Engineering and Applications*, vol. 6, pp. 196–206, 2013.

[14] H. Kim, J. Smith, and K. G. Shin, "Detecting energy-greedy anomalies and mobile malware variants," in *Proceedings of the 6th International Conference on Mobile Systems, Applications, and Services*, 2008, pp. 239–252.

[15] M. Masud, L. Khan, and B. Thuraisingham, *Data Mining Tools for Malware Detection*. Auerbach Publications and CRC Press, 2011.

[16] U. Orhan, M. Hekim, and M. Ozer, "EEG signals classification using the K-means clustering and a multilayer perceptron neural network model," *International Journal of Advanced Research in Computer and Communication Engineering*, vol. 38, pp. 13 475–13 481, 2011.

[17] D. M. W. Powers, "Evaluation: From precision, recall and f-factor to ROC, informedness, markedness and correlation," School of Informatics and Engineering, Flinders University, Adelaide, Australia, Tech. Rep., 2007.

[18] A. A. Sebyala, T. Olukemi, and L. Sacks, "Active platform security through intrusion detection using naive Bayesian network for anomaly detection," in *Proceedings of the London Communications Symposium*, 2002, pp. 1–5.

[19] A. Shabtai, U. Kanonov, Y. Elovici, C. Glezer, and Y. Weiss, ""Andromaly": A behavioral malware detection framework for android devices," *Journal of Intelligent Information Systems*, vol. 38, pp. 161–190, 2012.

[20] Z. Shihong, J. Zhang, and X. Lin, "An effective behavior-based Android malware detection system," *Security and Communication Networks*, vol. 8, pp. 2079–2089, 2015.

[21] S. Vijayarani and M. Muthulakshmi, "Comparative analysis of bayes and lazy classification algorithms," *International Journal of Advanced Research in Computer and Communication Engineering*, vol. 2, pp. 3118–3124, 2013.

[22] A. Widodo and B.-S. Yang, "Support vector machine in machine condition monitoring and fault diagnosis," *Mechanical Systems and Signal Processing*, vol. 21, pp. 2560–2574, 2007.

[23] D.-J. Wu, C.-H. Mao, T.-E. Wei, H.-M. Lee, and K.-P. Wu, "Droidmat: Android malware detection through manifest and API calls tracing," in *Proceedings of the 2012 Seventh IEEE Asia Joint Conference on Information Security*, 2012, pp. 62–69.

[24] H. Wu, S. Schwab, and R. L. Peckham, "Signature based network intrusion detection system and method," 2008, US Patent 7,424,744.

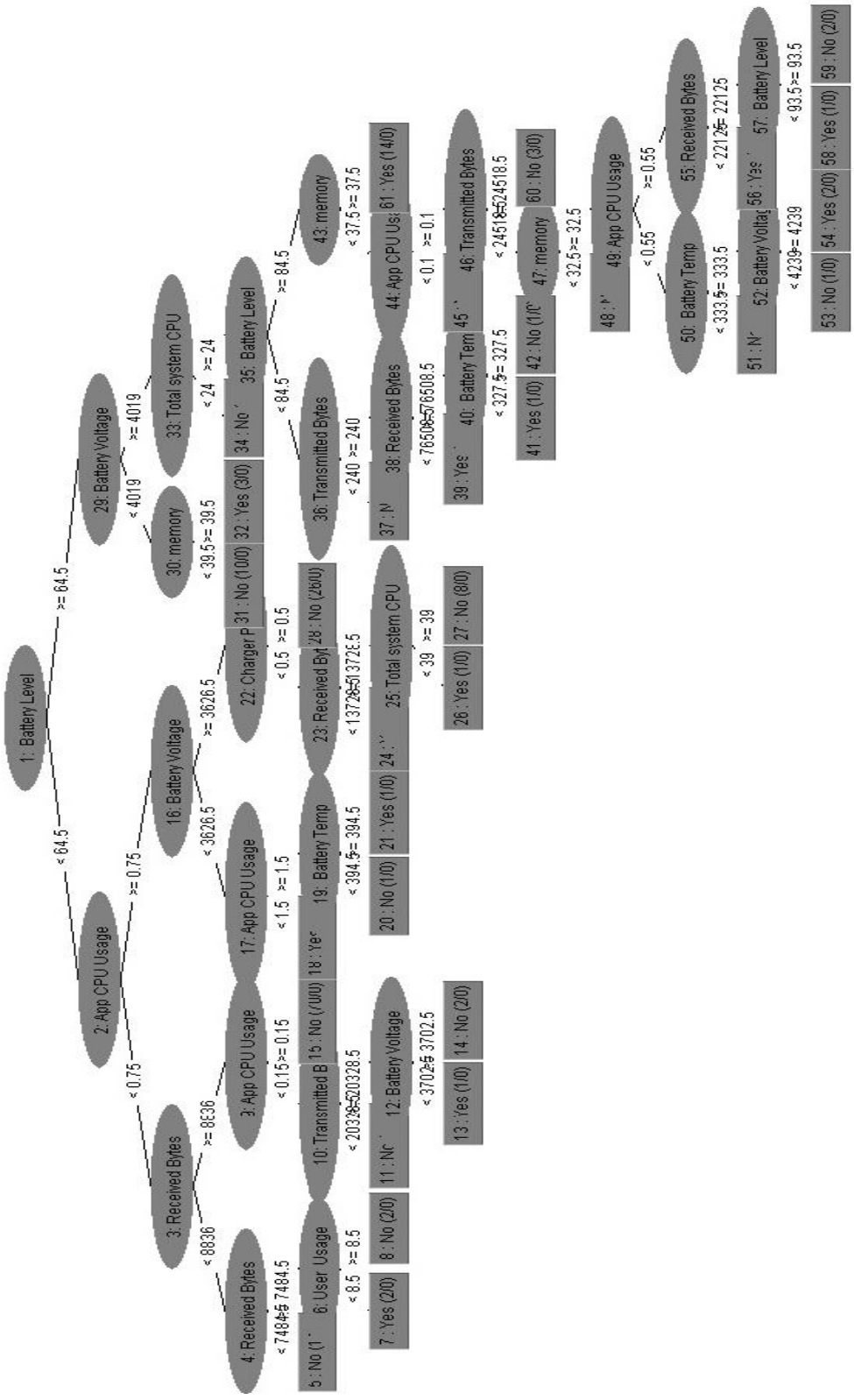

Fig. 2. Example of the random forest classifier.