Journal of Engineering Science and Technology
Vol. 14, No. 3 (2019) 1572 - 1586

A COMPARISON OF MACHINE LEARNING TECHNIQUES FOR ANDROID MALWARE DETECTION USING APACHE SPARK

LARAIB U. MEMON, NARMEEN Z. BAWANY*, JAWWAD A. SHAMSI

Systems Research Laboratory, National University of
Computer and Emerging Sciences, Karachi, Pakistan
*Corresponding Author: narmeen.bawany@nu.edu.pk

Abstract

Wide-scale popularity of Android devices has necessitated the need of having effective means for detection of malicious applications. Machine learning based classification of android applications require training and testing on a large dataset. Motivated by these needs, we provide extensive evaluation of Android applications for classification to either benign or malware applications. Using a 17-node Apache Spark cluster, we utilized seven different machine learning classifiers and applied them on the SherLock dataset - one of the largest available dataset for malware detection of Android applications. From the dataset of 83 attributes, we identified 29 suitable features of applications which are related in identifying a malware. Our analysis revealed that gradient boosted trees classification mechanism provides highest precision and accuracy and lowest false positive rate in detection of malware applications. We also applied our model to develop a real-time cloud based malware detection system. This research is novel and beneficial in providing extensive evaluation using large dataset.

Keywords: Android malware, Android security, Big data, Cloud, Malicious android applications, Malware detection, Spark.

1. Introduction

Detection of Android-based malware has been a major concern for the research community [1, 2]. Android is based on an open developer model, where applications can be developed and made available easily. This flexibility has led to increased popularity and usage. However, this model can also lead to availability and deployment of malicious applications [3].

A malware application can target an Android phone in numerous ways, for instance, it can lead to a privilege escalation [4] attack using which an intruder can get access to the phone through a backdoor. A malicious application can also get access to private data such as contacts and call logs [5, 6]. Further, it can consume large amount of resources on a phone which in turn causes starvation of resources for benign applications. In addition, possibilities of other attacks such as phishing, ransomware, and drive-by-download also exist [7, 8]. Table 1 lists a few major types of malware for smart phones.

Table 1. Types of malware and their harmful effects.

Malware type	Actions
Trojan/viruses	Infects device by attaching itself to a harmless application. Cause simple annoyance like advertisements and battery drainage. Can use device to make illegal transactions and steal data.
Phishing applications	Fraudulent apps disguising themselves as genuine to gain access to user's credit card information.
Botnets	Smart programs controlled by a "bot herder" that can stay idle waiting for certain commands by its controller. They keep providing the attacker with information and perform destructive actions on receiving the command.
Adware	Intermittently displaying ads and popping them up as urgent notifications. Bombarding the user with several pop ups that take the user to re-directional web pages. Ads are displayed aggressively so that the attacker can earn money.
Spying tool	Track user actions, locations, messages, call logs, recording videos and voice secretly. Broadcasts activities back to the hacker.
Auto rooter/root access hijacker	This automatically roots the device and hides itself in the system/core directories making it impossible to be removed from some devices. Once installed, it restricts out of its sandbox i.e. secure shell made by Android and takes control of the device.
Data stealer	The personal information, pictures, messages are stolen and used for personal gain by the attacker. This may be sold off to a third party or can be leveraged by the attacker for money.
Premium service abuser	Premium version of the application is purchased without user's awareness by violating the user permissions, through a back door or gaining the user's permission by a fake update.
Click fraudster	Misleading pop ups are displayed while seeking user's approval to gain more access rights or permissions, like close button for an ad would in fact give the app access to the camera roll, etc.
Malicious downloader	This malware keeps downloading other malicious applications on the device or will download large hidden files to exhaust user's storage.
Ransomware	Similar to data stealer. Can also hijack the phone, lock it, or render it useless unless the victim pays the attacker to gain the access back.
Power-off hijack	This malware hijacks the phone's shutdown process. This will launch a replica screen and makes the user believe that he has shut off the device. But the device continues to run with the screen off, sending messages and making calls from the device while spying on the user.

Conventional techniques for malware detection are based on either analysing malicious patterns in an application before installation (static analysis) or detecting them during execution (dynamic analysis). A hybrid approach is also possible which

relies on Machine Learning (ML) techniques to identify patterns for benign and malicious applications. A fundamental requirement of hybrid techniques is training and testing on a large dataset, which can lead to improved accuracy. However, existing techniques for machine learning based approaches suffer with low accuracy and high false positive rate (FPR). Also, these techniques have been tested on small datasets. With the emergence of big data [9] and increasing number of malware patterns [4], it is essential that machine learning based malware detection techniques be compared and tested on a large dataset to obtain high accuracy and low FPR.

In this paper, we are motivated to improve Android malware detection techniques using big data analytics. To this end, we provide a comparison of seven different ML classifiers on the SherLock dataset [10] - one of the largest available dataset on Android malware. Using 35 GB dataset and a 17-node Spark cluster, we compared different classifiers including Isotonic Regression, Decision Trees, Random Forest, Gradient Boosted Trees, Support Vector Machine (SVM), Logistic Regression, and Multilayer Perceptron. We observed that in general, tree based techniques provide better results. Specifically, Gradient boosted trees provide approximately 91% precision and accuracy - the highest among all the seven techniques. We also compared FPR in detection of benign applications and observed that the gradient boosted tree technique has the lowest false positive as well.

We also deployed our trained model on a private cloud to facilitate detection of malware applications in real-time. We envisioned that such a service, could be extremely useful for the community.

In summary, following are the major contributions of this paper:

- We provide extensive comparison of seven ML classifiers for detection of malware and benign applications using SherLock dataset for malware detection - one of the largest and most extensive dataset for android applications.
- From the list of 83 features in the original dataset we identified 29 features that are more significant in identifying the malware. These features were selected on the basis of p-value using chi-square method for hypothesis evaluation. (See *Appendix A*)
- We demonstrated through experiments that our results are improved in terms of classifier accuracy and reduced FPR.
- We integrated our model with a cloud to develop an online real-time anti-malware service. The service can be used by users to detect malicious applications on Android.

We anticipate that our research is highly beneficial in detection and prevention of Android malware. The remainder of this paper is organized as follows. Section II presents background and related work on the topic. Section III thoroughly discusses the methodology and experiments, followed by Section IV that present the results. Conclusion and future directions for research are elaborated in Section V.

2. Background and Related Work

Majority of the Android malware are multi-functional Trojans capable of stealing personal data stored on the user's phone. Such application pose a grave threat to the privacy and security of users [11, 12]. Malware are capable of exploiting permission to track users' location, photos, call log, messages and device's IMEI number. This

data may either be sold to cybercriminals or used by the attacker himself in harming the user. For instance, a malicious Android application called Porn Droid locks the user's phone and changes its access PIN number, to demand a $500 ransom. Similarly, different types of malware like spyware, ransomware, adware, rootkits, Trojans, viruses, botnets, and worms exist, which may harm a victim in different forms [13, 14]. Table 1 provides a summary of most popular malware.

Malware detection schemes in Android are mainly divided in two main branches as shown in Table 2. The static methods deal mainly in identifying a malware before installation. These include signature based detection, permission based detection, and bytecode analysis. Dynamic methods identify malware based on the patterns that are collected only after the application has been installed. It deals with anomaly analysis, taint analysis, and emulation based systems. However, existing malware detection methods are now being easily bypassed by astute developers using techniques like encryption, re-packaging and code obfuscation [14, 15].

Table 2. Android malware detection methods.

Malware detection types	Detection methods	Limitations
Static	Signature based detection	Detects malware from the signature of pre-existing malware, thus cannot detect new/unknown malware
	Permission based detection	Can classify a benign application as malicious due to very small difference in permission, hence high FPR
	Dalvik by code detection	High computation required on the device thus draining battery and exhausting memory
Dynamic	Anomaly based detection	Can classify a benign application as malicious if a benign application shows some red flags, like more battery consumption, high traffic usage, and hence high FPR.
	Taint analysis	High computation on the device. Takes too much time therefore not suitable for real-time detections. Slows down the device up to 20 times
	Emulation based detection	High usage of device's memory and resources. Device begins to lag
Hybrid	Machine learning approach	High training costs, high FPR due to dataset issues, Models not trained on latest data , slow detection

In addition to the above mentioned types , hybrid schemes also exist, which relies on combination of both static and dynamic methods [16]. The Hybrid methods utilize ML techniques for classification to improve the accuracy malware detection [17-19].

With the use of ML, activities of both malicious and benign applications are described with a set of features in the form of a vector representing the state of the device for, e.g., power consumption, battery usage, access privilege etc. ML algorithms are trained using feature vectors of known behaviours (malicious/benign) in order to correctly classify the unknown feature vectors [17]. Shabtai et al. [20] proposed a host-based malware detection system called Andromaly, that inspects the set of features such as, outgoing and incoming communication statistics, network

statistics, and battery usage. ML based anomaly detectors are used to classify the collected data as benign or malicious. Bente et al. [21] use similar features as Andromaly and add context and trust information. Similarly, Dini et al. [22] also include system calls feature. Haung et al. [23] used ML algorithms for detection of malicious applications using a permission based approach in which requested permissions were compared with the required ones. Different labelling techniques were applied based on the source of the application and methods for anti-virus classification. . The authors utilized Decision Tree, Naive Bayes, Ada Boost, and Support Vector Machine algorithms to detect malware. The results were not satisfactory as Naive Bayes produced inaccurate results, whereas Ada Boost classified all applications as benign. Sanz et al. [24] adopted a similar approach, i.e., comparing permissions and using only 357 benign and 249 malicious applications. Their results contained high FPR that were inadequate for efficient malware detection in Android. The machine learning algorithms used here were SimpleLogistic, Naïve Bayes, BayesNet, SMO, IBK, J48, RandomTree and RandomForest. The use of Bayesian Networks [25] with a small dataset showed some promising results.

Sahs and Khan [26] employed the use of SVM with modified kernels approach. Their results were also not promising and had a high FPR in detecting Android malware. In a recent study by Nancy [27], decision trees on signature based malware detection showed 90% accuracy. However, this approach relied on identification by matching the signature of functions with pre-existing collection of malicious signatures. This caused failures when signature was obfuscated or a new malware signature not existing in the database was used.

Table 3 summarizes existing work on Android malware detection using ML. Many existing machine learning based solutions have high FPR ranging from 15.86% to 33.79% [23]. This is typically due to over fitting of data in the testing set, resulting in a model that floods the user with false notifications. Experiments in prevailing solutions were conducted using small dataset and a few features were used to train the models. Many researchers have worked on Genome dataset which is not par with the todays' Android malware [25-27]. This dataset has a collection of malware samples collected from August 2010 to October 2011. Hence, Genome dataset does not include emerging and more recent mobile malware samples. Therefore, we used SherLock dataset that comprises of latest malware samples and contains over 600 billion data points in over 10 billion data records. The focus of this research is to compare various ML techniques utilizing large dataset for training and testing, such that, improved results can be achieved.

In the next section we describe the methodology and details of experiments that were conducted to devise a model for malware detection.

Table 3. Summary of research work on ML based Android malware detection.

Paper	ML Technique and Data used	Results
Amos et al. [17]	- 1330 malicious, 408 benign applications used (APKS) - Logistic Regression - Bayes Net - Naïve Bayes	- Bayes net best with 81.25% correct classification but with high FPR i.e. 31% - Logistic regression worst with 68.75% detection - Classifiers with good

	- Random forest - Multilayer Perceptron	classification, i.e., >78% had high FPR rates, i.e., 15-33% - Cross validation with actual data did not reveal same results due to small sample size and over estimation
Shabtai et al. [20]	- Naïve Bayesian - Random forest - logistic regression - SVM: Support Vector Machine.	- Random Forest performed best with 0.001 FPR and 0.99 TPR - SVM had worst results with mean FPR of 0.20 among different types of applications - Data from only 6 applications used
Sahs and Khan [26]	Support Vector Machine on Scikit-learn framework using Weisfeiler-Lehman kernel. Data trained to classify all applications as benign so that all other behaviors can be classified as malicious. Permission based classification - benign only dataset for training	- Bit vector kernel correctly classifies malware - High FPR for benign applications that request same permissions as malware - String kernel classified everything as malware - Graph kernel shows FPR > TNR
Shamili et al. [28]	Dataset of symbian OS (not widely used today) - SVM	- 80-90% malware detection rates based on number of contaminated SMSs
Nancy [27]	- Bayesian classification using APK analyser by reading signatures	With maximum features used: - TPR = 0.906 - FPR = 0.094 - Accuracy = 0.921 - Error = 0.079 Thus acceptable results
Huang et al. [23]	- Ada Boost - Naive Bayes - Decision Trees - SVM	- Poor detection due to highly imbalanced dataset with only 0.004% malicious applications - Ada boost identified all apps as normal - Naive Bayes produced imprecise results - Decision trees and SVM produced precise results
Sanz et al. [24]	- Simple Logistic - Naive Bayes - Bayes Net - SMO - IBK - Decision Trees - Random Tree	- High FPR i.e. up to 27% - Bayes net had TPR of 45% - Other classifiers had TPR ranging from 87-92%

	- Random Forest	
Peiravian and Zhu [29]	- 1260 malware, 1250 benign apps - APK and manifest files used for permission based comparison - SVM - Decision Trees - Naïve Bayes - Bagging Vectors	- Accuracy of classifiers between 93-96% - Precision between 92-94% - Recall between 86-94% This showed acceptable results
Narudin et al. [16]	Network traffic from 1400 applications - Random forest - Multi layer perceptron (MLP) - Naïve bayes - Bayesian networks - KNN	- Random Forest showed 99.9% detection rate with Genome Dataset(release in 2013) - KNN gave 84.6% accuracy
Amos et al. [30]	Network traffic activity of 147 malware, 49 benign - Decision trees	- 90% accuracy - Fails on encrypted network traffic/traffic obfuscation

3. Methodology

The major goal of the research is to compare different ML based classification schemes in order to assess their efficacy in determining malware. For performance comparison, we aimed to run our experiments on a large dataset, that is, the SherLock dataset.

We conducted our experiments on a Spark cluster consisting of 17 nodes. Each node had a pre-compiled version of Spark installed. Out of these, 16 nodes were configured as slave nodes. Each node is a Quad core machine with Intel core i5 processor and 8GB RAM. A Network File System (NFS) cluster was created to facilitate data access across the worker nodes such that each worker node was able to access the partition allocated to it by the driver program. The dataset used for the experiments was obtained from the Ben-Gurion University of the Negev's Information Systems Engineering Department via BGU Cyber Security Research Center [6]. The dataset contained 10 billion data records from 50 users.

Table 4 provides description of the files used. Due to constrained resources, we limited dataset to Quarter 3 of 2016 with 35GB size. After merging Moriarty and T4 and creating labelled points, the file selected for experiments had 34.23 million records. The data exhibits a class imbalance because benign samples are only 1 percent as shown in Table 5.

To rectify this imbalance, equal data points of positive and negative were selected. Dataset was divided with a ratio of 80:20 for training and testing the classifiers, respectively.

Table 4. Files selected for use.

File Name	Description
T4	Sampled every 5 seconds Network traffic (bytes/packets) CPU utilization of each core + aggregated Memory, cache, virtual memory Number of processes in user mode, kernel mode Number of threads created Processes running, processes blocked Internal, external storage
Moriarty	Sampled when clue is recorded by the mortality malicious agent Action, action type, behaviour (malicious, benign)

Table 5. Distribution of data.

Action Label	Count	Fraction
1 - Malicious	33864838	0.989275
0 - Benign	367148	0.010725
Total	34231986	1

Creating Labelled points

Labelled points were created by merging data from Moriarty and T4. Only those T4 instances were taken which were collected during the Moriarty application session having both benign and malicious data. Next, this was joined to the data frame on the basis of user, start and end timestamp and session. This provided the complete information about each of the T4 record with respect to the type of actions/sessions that was being carried out by Moriarty when those observations were being made.

We applied seven different ML techniques to identify the most appropriate classification method for our model. These include Logistic Regression, Decision Tree, Random Forest, Gradient-boosted Tree, Multilayer Perceptron, SVM and Isotonic Regression. We initially used all 83 features to train our model. However, it took more than 6 hours to train the dataset. We noticed that all features were not significant in detecting the malware. In order to determine relevant features, we applied Chi-Square method to obtain p-values. Attributes with p-value > 0.05 were discarded. In total, 29 features were selected out of total 83 features. After reducing the features, the training time was reduced to 90 minutes.

For performance comparison, we utilized five metrics, that is, Precision, Recall, F1 score, Accuracy, and false positive rate [31-33]. Table 6 lists the details of the performance metrics. Along with class-wise metrics, the label-wise results are also presented to get better hold of class-wise performance. As explained earlier, skewness of the data was fixed by selecting equal data points for negative and positive. This enabled better identification of both malicious and normal applications, and reduced FPR.

Table 6. Performance evaluation metrics.

Weighted	Label Wise
Precision	Precision
Recall	Recall
F1 Score	F1 Score
Accuracy	
FPR	

4. Results and Discussion

We applied the seven listed classifiers on the dataset and assess the performance using five different metrics. We observed that among all the seven schemes, lowest FPR of 9.2% was produced by Gradient boosted Trees. This was lower than the FPR achieved by other researchers [10, 14, 19]. Moreover, for gradient boosted trees, classification of both benign and malicious labels produced better results in terms of precision i.e. 94% and 85%, respectively. We anticipate that this is due to the regularization nature of gradient boosted trees which prevents overfitting and reduces error.

We also observed that Support Vector Machine (SVM) did not performed well for binary classification giving least accuracy and highest FPR. This was expected because SVM is not suitable for highly skewed/imbalanced dataset. SVM does not perform well when there are many data points. This is because of the fact that SVM uses kernel tricks in which the number of parameter increase with number of data points. Performance of both types of regression, i.e., Logistic and Isotonic were not also up to the mark.

Although the skewness of the data was fixed, the results from these were still off by 10% to 15% compared to the gradient boosted trees. Multilayer Perceptron produced better results compared to the Isotonic Regression, Logistic Regression, and SVM.

However, Gradient boosted Tree outperformed other techniques in all the categories with F1 score of 0.9. SVM had the worst performance with 0.49 F1 score. It is worth mentioning that by this point skewness of the data had been fixed, therefore the weighted results are not biased over any class. In general, tree based methods, i.e., gradient boosted tree, decision trees and random forest were among top performers. The detail of results are given in Table 7.

Table 7. Experiment results weighted.

	Precision	Recall	F1 Score	Accuracy	FPR
Isotonic Regression	0.788365	0.78841	0.773016	0.78841	0.357233
Random Forest	0.899371	0.896722	0.896651	0.896722	0.101646
Decision Trees	0.902023	0.897264	0.897371	0.897264	0.096595
Gradient-boosted trees	0.913962	0.902459	0.908971	0.902459	0.092702
Multilayer perceptron	0.802102	0.805423	0.797682	0.805423	0.302645
SVM	0.719654	0.629325	0.49195	0.629325	0.61636
Logistic Regression	0.698069	0.700145	0.689902	0.700145	0.366314

One of the major objective of this research was to reduce the FPR of the classifiers. Though, the existing work had been good in recognizing malicious software, they suffer with high FPR. Using a solution with high FPR is not feasible as it will yield too many false notifications. Therefore, reliability of such a solution remains questionable and a user may not trust the results.

In our model, FPR was successfully reduced to 9.3% in gradient boosted trees and 9.8% and 9.6% with random forest and decision trees, respectively. Figure 1 illustrates the results. In general, FPR of trees/forest algorithms were much lower than regression and SVM algorithms.

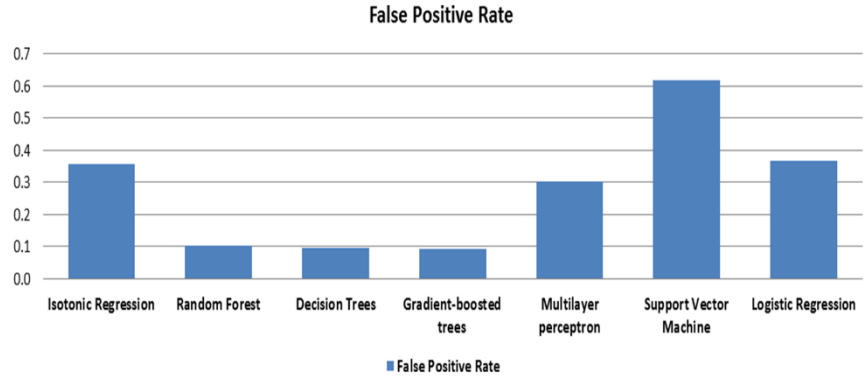

Fig. 1. False positive rate.

To ensure that the classifiers work equally well for the detection of both malicious and benign applications, we conducted label wise experiments. Figures 2 and 3 show the results. For benign applications, SVM has the lowest precision. The trade-off between precision and recall is evident from the results. Gradient boosted trees show the best precision, that is, 95%. It is to be noted that SVM produced the only acceptable result here i.e. its recall for benign class, which was 99%. For benign the 0.9 F1 score of gradient boosted and decision trees and 0.89 of random forest, indicates that they have performed equally well in this classification. These results are significantly important as many of the previous classifiers though are good at detecting malicious applications, they falsely suggest benign as being malicious, thereby increasing the false positive.

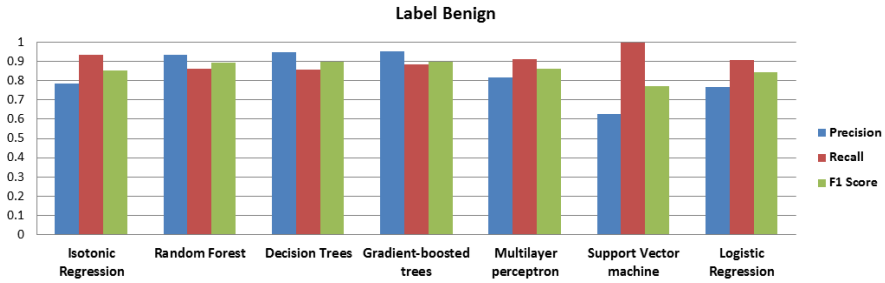

Fig. 2. Label benign.

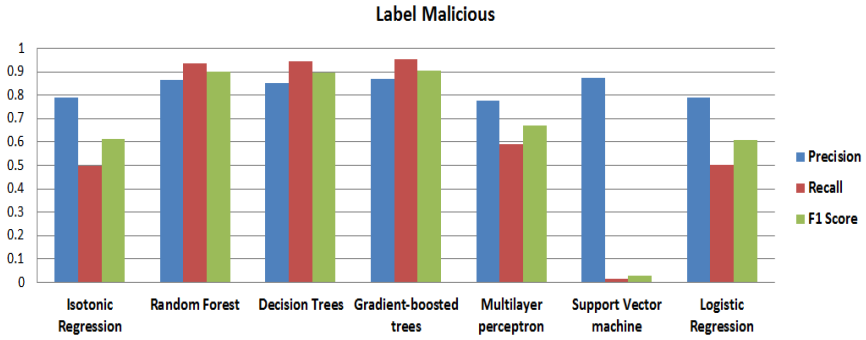

Fig. 3. Label malicious.

We also incorporated the model generated from gradient boosted trees to create a malware detection service that was deployed on a private cloud. The application takes input of feature vectors that as tracked from the smart phone and passes them to the model to classify an application as either malicious or benign as shown in Fig. 4.

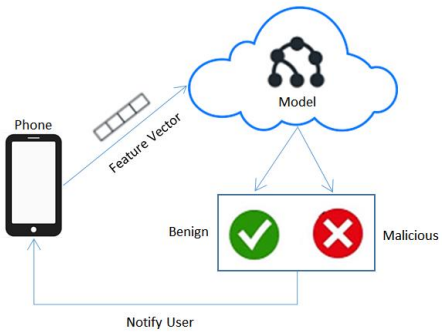

Fig. 4. System architecture.

5. Conclusion and Future Work

Malware detection on Android is significant and challenging. We developed an effective solution for this problem. We utilized seven ML classifiers to classify malicious and benign applications. Using 35 GB subset of SherLock dataset, we observed that gradient boosted trees with 90% overall accuracy, recall, and F1 score and 91% precision, have been most accurate in detecting malicious applications. This technique also had lowest FPR of 9% only. The final results are based on the features selected through p-values, but it was noticed that increasing or decreasing the number of features lead to adverse effect on the results. We carefully selected the features that were more significant in detecting malware.

The model was incorporated into a malware detection system that was tested on a private cloud with Apache Spark, where set of feature vectors were classified as malware or benign in real time. In order to better utilize resources and to avoid memory overflow, files were also read in partitions.

As a future work, we plan to conduct experiments and verify the model on the entire dataset. Further, we intend to integrate the model with a public cloud to design an effective real-time solution.

Acknowledgement

This research is supported by Higher Education Commission, Pakistan grant HEC NRPU 5946 - 2016.

References

1. Chen, L.; Hou, S.; Ye, Y.; and Chen, L. (2017). An adversarial machine learning model against android malware evasion attacks. *Lecture Notes in Computer Science (LNCS) book series*, 10612.
2. Sufatrio; Tan, D.J.J.; Chua, T.; and Thing, V.L.L. (2015). Securing Android : A survey , taxonomy , and challenges. *ACM Computing Surveys (CSUR)*, 47(4), 45 pages.
3. Arshad, S; Shah; M.A; Wahid, A; Mehmood, A; Song, H.; and Yu,.H. (2018). SAMADroid: A novel 3-level hybrid malware detection model for Android operating system. *IEEE Access*, 6, 4321-4339.
4. Niazi, R.H.; Shamsi, J.A.; Waseem, T.; and Khan, M.M. (2016). Signature-based detection of privilege-escalation attacks on Android. *Proceedings of the Conference on Information Assurance and Cyber Security (CIACS)*. Rawalpindi, Pakistan, 44-49.
5. Mazlan N.H.; and Hamid, I.R.A. (2018). Using weighted based feature selection technique for Android malware detection. *Mobile and Wireless Technologies 2017*, 54-64.
6. Garcia, J.; Hammad, M.; and Malek, S.; (2015). Lightweight, obfuscation-resilient detection and family identification of Android malware. *ACM Transactions on Software Engineering and Methodology*, 9(4), Article 39, 27 pages.
7. Chen, J.; Wang, C.; Zhao, Z.; Chen, K.; Du, R.; and Ahn, G.-J. (2017). Uncovering the face of Android ransomware: characterization and real-time detection. *IEEE Transactions on Information Forensics and Security*, 13(5), 1286-1300.
8. Dini, G.; Martinelli, F.; Matteucci, I.; Petrocchi, M. Saracino, A.; and Sgandurra, D. (2015). Risk analysis of Android applications: A user-centric solution. *Future Generation Computer Systems*, 80, 505-518.
9. Shamsi, J.; Khojaye, M.A.; and Qasmi, M.A. (2013). Data-intensive cloud computing: Requirements, expectations, challenges, and solutions. *Journal of Grid Computing*, 11(2), 281-310.
10. Mirsky, Y.; Shabtai, A.; Rokach, L.; Shapira, B.; and Elovici, Y. (2016). SherLock vs. Moriarty: A smartphone dataset for Cybersecurity Research. *Proceedings of the ACM Workshop on Artificial Intelligence and Security*. Vienna, Austria, 1-12.
11. Felt, A.P.; Finifter, M.;Chin, E; Hanna, S; and Wagner, D. (2011). A survey of mobile malware in the wild. *Proceedings of the 1st Workshop on Security and Privacy in Smartphones and Mobile Device*. Chicago, Illinois, 3-14

12. Zhou, Y.; and Jiang, X. (2012). Dissecting Android malware: Characterization and evolution. *Proceedings of the IEEE Symposium on Security and Privacy. San Francisco, California*, 95-109.
13. Lindorfer, M.; Neugschwandtner, M.; Weichselbaum, L.; Fratantonio, Y.; van der Veen, V.; and Platzer, C. (2016). ANDRUBIS - 1,000,000 apps later: A view on current android malware behaviors. *Proceedings 3rd International Workshop on Building Analysis Datasets and Gathering Experience Returns for Security (BADGERS). Wroclaw, Poland*, 3-17.
14. Arshad, S.; Khan, A.; Shah, M.A.; and Ahmed, M. (2016). Android malware detection and protection: A survey. *International Journal of Advanced Computer Science and Applications (IJACSA)*, 7(2), 463-475.
15. Freiling, F.C.; Protsenko, M.; and Zhuang, Y. (2015). An empirical evaluation of software obfuscation techniques applied to android APKs. *Lecture Notes of the Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering (LNICST)*, 153.
16. Narudin, F.A.; Feizollah, A.; and Ghani, A. (2014). Evaluation of machine learning classifiers for mobile malware detection. *Soft Computing*, 20(1), 343-357.
17. Amos, B.; Turner, H.; and White, J. (2013). Applying machine learning classifiers to dynamic Android malware detection at scale. *Proceedings of the 9th International Wireless Communications and Mobile Computing Conference (IWCMC)*. Sardinia, Italy, 1666-1671.
18. Malik, S.; and Khatter, K. (2018). Malicious application detection and classification system for Android mobiles. *International Journal of Ambient Computing and Intelligence*, 9(1), 20 pages.
19. Zulkifli, A.; Hamid, I.R.A.; Shah, W.M., and Abdullah, Z. (2018). Android malware detection based on network traffic using decision tree algorithm. *Recent Advances on Soft Computing and Data Mining*, 485-494.
20. Shabtai, A.; Kanonov, U.; Elovici, Y.; Glezer, C.; and Weiss, Y. (2012). "Andromaly": A behavioral malware detection framework for android devices. *Journal of Intelligent Information Systems*, 161-190.
21. Bente, I.; Hellmann, B.; Vieweg, J.; von Helden, J.; and Dreo, G. (2012) TCADS: Trustworthy, context-related anomaly detection for smartphones. *Proceedings of the 15th International Conference on Network-Based Information Systems*. Melbourne, Victoria, Australia, 247-254.
22. Dini, G.; Martinelli, F.; Saracino, A.; and Sgandurra, D. (2012). MADAM: A multi-level anomaly detector for Android malware. *Lecture Notes in Computer Science* (LNCS), 7531, 240-253.
23. Huang, C.-Y.; Tsai, Y.-T.; and Hsu, C.-H. (2013). Performance evaluation on permission-based detection for Android malware. *Smart Innovation, Systems and Technologies (SIST)*, 21.
24. Sanz, B.; Santos, I.; Laorden, C.; Ugarte-Pedrero, X.; Bringas, P.G.; and Alvarez, G. (2013). PUMA: Permission usage to detect malware in Android. *Advances in Intelligent Systems and Computing (AISC)*, 189.
25. Yerima, S.Y.; Sezer, S.; McWilliams, G.; and Muttik, I. (2016). A new Android malware detection approach using Bayesian classification. *Proceedings of the 27th International Conference on Advanced Information Networking and Application (AINA)*. Barcelona, Spain, 121-128.

26. Sahs, J.; and Khan, L. (2012). A machine learning approach to Android malware detection. *Proceedings of the European Intelligence and Security Informatics Conference (EISIC).* Odense, Denmark, 141-147.
27. Nancy, D.S. (2016). Android malware detection using decision trees and network traffic. *International Journal of Computer Science and Information Technologies,* 7(4), 1970-1974.
28. Shamili, A.S.; Bauckhage, C.; and Alpcan, T. (2010). Malware detection on mobile devices using distributed machine learning. *Proceedings of the 20th International Conference on Pattern Recognition*. Istanbul, Turkey, 4348-4351.
29. Peiravian, N.; and Zhu, X. (2013). Machine learning for Android malware detection using permission and API calls. *Proceedings of the International Conference on Tools with Artificial Intelligence (ICTAI). Hendorn, Virginia, United States of America,* 300-305.
30. Garcia, B.T. (2015). A framework for detection of malicious software in Android handheld systems using machine learning techniques. *Master Program in Security of Information and Communication Technologies (MISTIC).* Open University of Catalonia, Barcelona, Spain.
31. Sreeram, I.; and Vuppala, V.P.K. (2017). HTTP flood attack detection in application layer using machine learning metrics and bio inspired bat algorithm. *Applied Computing and Informatics,* 15(1), 59-66.
32. Jia, B.; Huang, X. Liu, R.; and Ma, Y. (2017). A DDoS attack detection method based on hybrid heterogeneous multiclassifier ensemble learning. *Journal of Electrical and Computer Engineering,* Article ID 4975343, 9 pages.
33. Williams, N.; Zander, S.; and Armitage, G. (2006). A preliminary performance comparison of five machine learning algorithms for practical IP traffic flow classification. *ACM SIGCOMM Computer Communication Review,* 36(5), 7-15.

Appendix A

List of Features Used

1. CpuHertz
2. CPU_0
3. CPU_1
4. CPU_2
5. CPU_3
6. Traffic_MobileTxBytes
7. Traffic_MobileTxPackets
8. Traffic_TotalRxPackets
9. Traffic_TotalTxBytes
10. Traffic_TotalWifiRxPackets
11. Traffic_TotalWifiTxPackets
12. Battery_current_avg
13. Battery_level
14. Battery_temperature
15. Battery_voltage
16. SwapCached
17. Shmem
18. Slab
19. SReclaimable
20. SUnreclaim
21. KernelStack
22. VmallocUsed
23. VmallocChunk
24. wcd9xxx_cpu0
25. pn547_cpu0
26. SLIMBUS_cpu0
27. SLIMBUS_sum_cpu123
28. tot_irq
29. procs_running