Android Malware Detection using Machine Learning

Atika Gupta, Sudhanshu Maurya, Divya Kapil, Nidhi Mehra, Harendra Singh Negi

Abstract: ***Machine Learning is empowering many aspects of day-to-day lives from filtering the content on social networks to suggestions of products that we may be looking for. This technology focuses on taking objects as image input to find new observations or show items based on user interest. The major discussion here is the Machine Learning techniques where we use supervised learning where the computer learns by the input data/training data and predict result based on experience. We also discuss the machine learning algorithms: Naïve Bayes Classifier, K-Nearest Neighbor, Random Forest, Decision Tress, Boosted Trees, Support Vector Machine, and use these classifiers on a dataset Malgenome and Drebin which are the Android Malware Dataset. Android is an operating system that is gaining popularity these days and with a rise in demand of these devices the rise in Android Malware. The traditional techniques methods which were used to detect malware was unable to detect unknown applications. We have run this dataset on different machine learning classifiers and have recorded the results. The experiment result provides a comparative analysis that is based on performance, accuracy, and cost.***

Keywords: Android, Malware, Machine learning, Classifiers

I. INTRODUCTION

As we know that smartphones are the requirement of the new era and it has been around us and has become an indispensable part of our day-to-day lives. As we are getting several benefits, we are expecting more and more out of it. Due to this, our consumption of cell phones is increasing, so as our dependability, and we began to expect more and more from our device. Today, we are in need of smartphones, traditional phones are not sufficient to solve our daily real-time task application, but smartphones are designed to do so. Smartphones are categorized according to the OS installed in it. The most popular OS includes Android OS, iPhone OS, Blackberry RIM OS, and Microsoft Windows OS and out of which Android captures the market with 86.2% sales in 2018 [1]. With the increase in demand for these devices, there is a huge competition amongst manufacturers to deliver as many products as they can to earn maximum profit. In this sprint of producing more and more devices security somehow is compromised.

Revised Manuscript Received on September 25, 2019.

Atika Gupta, School of Computing, Graphic Era Hill University, Uttarakhand, India. E-mail: atika04591@gmail.com

Dr Sudhanshu Maurya, School of Computing, Graphic Era Hill University, Uttarakhand, India. E-mail: dr.sm0302@gmail.com

Divya Kapil, School of Computing, Graphic Era Hill University, Uttarakhand, India. E-mail: divya.k.rksh@gmail.com

Nidhi Mehra, School of Computing, Graphic Era Hill University, Uttarakhand, India. E-mail: nidhigehu@gmail.com

Harendra Singh Negi, Computer Application, Graphic Era Deemed to be University, Uttarakhand, India. E-mail: mail.harendrasinghnegi@gmail.com

There is a huge pool of android developers with which there is an alarming growth in the rate of malicious apps which is becoming an issue for concern. Android is considered vulnerable over iOS for the reason that it allows the apps to be installed from the sources which are unverified such as websites and apps from third-party stores [2]. These malicious apps can hamper security as they can steal confidential information and compromise the system. If we roughly calculate a malicious android app is up every 10 seconds [3]. Android is also amongst the most valuable target for the developers of malware. As developers are finding interest and gains in malware so we have around 49 Android malware [4]. Due to which a great amount of effort in coding the tools for software security which are capable of handling these continuously flourishing market of malware.

Machine Learning is the subset of AI, which gives an idea that if we feed correct data to the machine, it can learn problem-solving by itself with experience [5]. It can also be explained as the field which makes the computer learn based on its experiences rather than programmed. The machine is given the learning dataset, e.g. if we want the machine to learn how a bird looks like, then we have to provide it with the images of different types of birds from different angles so that the next time the machines sees a bird it recognizes that yes it is a bird and also record it for its experience [6].

Machine Learning classifiers are playing an important role in the development of bright systems. ML takes a dataset as input and produces a model that is capable of handling new data. Adopting such methodologies has always proven to enhance the accuracy of the detection system. Many other methods available in the market such as Anti-virus which drain the system's resources, we can use some other methodologies to detect the malware as we don't want to engage many resources and hamper the responsiveness. In this paper, we have used dataset CICAndMal2017 which is Android dataset and have implemented several machine learning classifiers such as Naïve Bayes Classifier, Random Forest, Decision Tress, Boosted Trees, K-Nearest Neighbor, and Support Vector Machine, on it to check the accuracy of each with respect to that particular dataset.

In this paper, our work is divided into the following sections. The discussion in Section II is about related Work. Sec III shows the architecture of android, section IV gives the idea of related vulnerabilities, section V describes the methodology of our work in which we discuss evaluation metrics and the classifiers which were used, section VI show the results obtained and discussion, And finally, the conclusion our paper in section VII.

II. RELATED WORK

There is a noteworthy work is done in the field of detection of Android Malware. The analysis is broadly categorized into two categories:

a) Static and
b) Dynamic.

Static malware detection is those which are done without running the app and can include 1) API calls 2) permissions which can be extracted from a special file in the package known as *AndroidManifest.xml*. On the other hand, Dynamic is the one which is done when the application is running which includes [7]:

a) Network traffic analysis
b) IP address
c) Battery usage etc.

Also, these two approaches can be mixed to generate a hybrid solution.

Uses a total of 30 apps and 5 malware samples namely: Gold Dream, DroidKungFu2, Angry Birds Rio Unlocker, Snake, PJApps. The resources were allocated before the app starts and behavioral patterns were extracted. The features used were all divided into seven categories: Network, Power, Process, CPU, SMS, Memory, and Virtual Memory, and these features were inputted to information gain to select the features. To this input data, four types of classifiers are applied: Naïve Bayesian, SVM-Support Vector Machine, Random Forest, and Logistic Regression. The author in this paper concludes that the performance of Random Forest proves the most promising [7]. The static and dynamic approaches are employed using two methods: *Heuristic method and Signature-based method* [8]. In signature-based, the common method used is an anti-virus vendor and it depends upon identifying a unique signature in the malware. But these methods fail when it comes to unknown malicious code. On the other side, heuristic depends upon the rules which are noted by either the specialist or by the machine learning classifiers that can define the suspicious behavior and can also detect the malicious unknown code [9]. In this paper [10][1], the author has extracted the Android APK file which is equivalent to jar files in java by reverse engineering, as on its extraction AndroidManifest.xml is accessible which contains all the permissions. Permissions are seen to check for the standard and non-standard ones and CFG (control flow graphs) are generated using the raw bytecodes.

Application permissions and experiments based on feature selection method is used in [9]. Feature Extractor (communicates with different components and extract the feature metrics), a processor (for analysis and detection), Threat weighing unit (collects the analysis result from each processor and apply the algorithm) and Main service (gets information from an alert manager about the malicious app and decides the action to be taken) modules are used. User rating, application permissions, the number of ratings given, size of the particular application are considered and the ML algorithms Random forest, Bayesian network, Decision tree are applied. The number of samples used was around 820 and the experiment concluded that a higher accuracy rate can be achieved without using the false positives rate. Some research paper presents the use of Neural Networks to analyses the category of the application using permissions from Manifest file by multilayered feedforward networks. This feedforward network has 2 layers, one hidden layer performs the sigmoid function and another output linear function is deployed. The author believes that the permission written in the manifest file can be modified by the malicious code authors, so to check this the author permuted the test data and fed the network [11]. One more research is based on the APK analyzer. Here an APK analyzer is applied to a totality of 6863 applications, out of which 3938 applications were benign and 2925 were malicious. Features like permissions, API calls, and many others were drowned out from the manifest and dex class file. All the features were combined using mutual information and the Bayesian algorithm was used for classification [12]. Another research work presents the features extracted from the manifest file such as application name, application category, description. Package, price, rating count, rating value A total of 18,174 applications were extracted for the features and were classified using K-Mean clustering. It was analyzed that these methods were enough to detect the malware, installation, and running was not required [13]. One more interesting research found that discusses using *TinyDroid* which uses Machine learning techniques and instruction simplification. It proposes a model that first extract the symbol-based opcode from the Android APK. In the second step it uses the N-gram approach and the classifier is trained to feed it to the machine learning classifiers. The Drebin Dataset is preprocessed to remove the unwanted features. This experiment shows that TinyDroid gets a higher precision rate and a less false alarm rate [14]. The study includes extracting the features from Android devices and fed the data to two machine learning algorithms which are the Bayesian algorithm and Bayesian algorithm with a Chi-square filtering test. The result of this experiment shows that the Bayesian algorithm with Chi-square gives a more accurate result which is near about 89% and the precision rate of Bayesian algorithm is almost 80% [15]. Another Study shows how to detect the fresh malware which is self-updating because these are the most dangerous ones which can steal confidential information. They have used 5-10 self-written Trojan malware which has two versions: one for the benign app and one for the malicious one. Several traffic-based information is extracted from the app while the app is in running state, and the apps are installed in the devices and the traffic is analyzed. This traffic analysis helps us to distinguish the benign and malicious app. The features are extracted and measured within equal time intervals. This study was successful to detect the malicious repacked app with the help of traffic analysis [16].

The lowest layer of the Android architecture which is Linux kernel is accessed to extract the Linux based features and afterward, these features are used to detect malicious applications. A total of 59 features were abstracted which includes: CPU, network, memory, etc. and on and all 6 malware was run and the system to observed to extract the above-mentioned features.

The data collection process is initiated every 10 secs and the collected data is sent to the server which in turn classifies this data. 39 features out of the total 59 were selected and the results were compared before and after the application of the features. Feature selection enhances the precision rate and lowers down the false-positive rate [17].

III. ANDROID ARCHITECTURE

The architecture of Android is depicted in Figure 1. It consists of the following layers:

a) *Application Software:* The first layer of this architecture is the application software layer, which has all the applications with which the user interacts. These apps provide a way for the user to control the hardware. Examples may include Camera, Clock, Calendar, and many more.

b) *Application Framework:* This layer is the second layer in the architecture which gives many services of higher level to applications that are in the form of java classes. Some major services such as Content providers, Activity manager, Notification manager, Resource manager, and view system are some of the components of this layer.

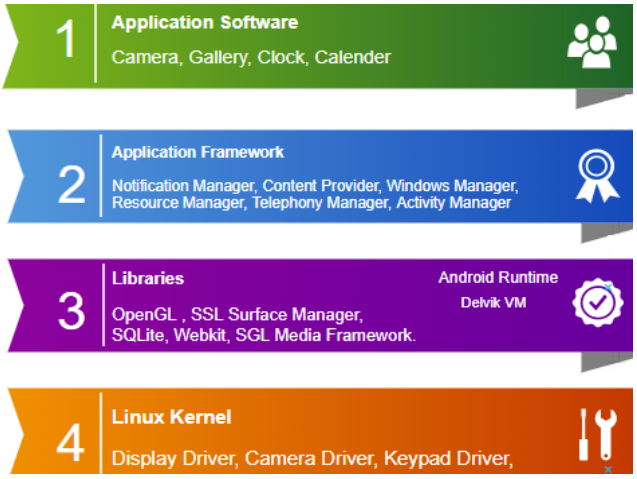

Figure 1: Layered Architecture of Android

c) *Libraries:* The third layer is the bundle of libraries which includes WebKit open-source web browsing engine, familiar library libc, Android built-in database SQLite which a convenient repository for dividing and storing of application data, libraries which contains features to record audio & video, libraries which are responsible for securing the SSL.

d) *Linux kernel Layerl:* The lowermost layer is the Linux kernel layer which includes almost 115 patches altogether. The layer provides an abstraction level between the hardware of the device and has all the drivers which are essential like keyboard, camera, display, etc. Here all the thing in whose handling Linux is good at is done by the kernel such as an extensive array of device drivers which provide us with the ease of hardware peripheral interfacing, the networking, etc.

IV. VULNERABILITIES

Here, the classification of each vulnerability is done in accordance with the layer from which it is generated. The purpose of this classification is to get an idea of the weak areas of the mobile architecture implementation.

a) *Application layer*: The major attack which occurs on this layer is through the browser where the attacker can execute some unwanted code to get into the system. After making the way into the system it can get access to all the sensitive data including the gallery. Also, the injection of cookies cannot be controlled.

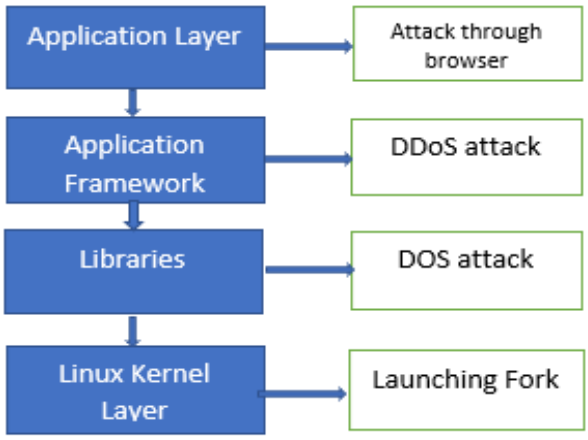

Figure 2: Mobile architecture and associated vulnerability

b) *Application Framework Layer*: The attack which takes place at this layer is the DDoS attack and can also be unauthorized access. In some cases, it has been seen that the attacker gets deactivated all the locks on sensitive data, which may compromise much sensitive information like the contacts, gallery, camera, etc.

c) *Library Layer:* The vulnerabilities here are having a huge impact. The attack launched was the DOS (Denial of service) attack which leads to the stack overflow by the execution of the API passing the wrong number of arguments. Some attacks were also carried out by the malware to get the authorization from the root of the device. By doing this, an attacker can change the code and make it act maliciously without anyone knowing about the change in the signed APK.

d) *Linux Kernel Layer:* This layer is the most secure, but still some of the vulnerabilities can be found. Any number of fork commands (command to create a process) an be launched without authenticating the identity of the source by using the wrong set of permissions [18].

V. METHODOLOGY

The investigation is carried out using two datasets, and the details of these datasets are depicted in table 1.

TABLE I. DATASETS AND THEIR DETAILS

Dataset	Samples	Features	Benign	Malware
Malgenome	3799	215	2539	1260
Drebin	15036	215	9476	5560

The first dataset which is used for this experiment is Malgenome dataset which consists of features from 3799 application samples, in which the number of apps with malware is 1260 and the benign applications are 2539. This dataset is widely used by the researchers and is obtained via static code analysis of the Android App. The second dataset which is used is Drebin dataset where the samples used were 15036, from which the number of malicious apps is 5560 and the benign apps were 9476. The number of features used in both the datasets is 215.

The flow of our experiment is depicted in figure 3. We started with the raw dataset of Android Malware. The data is then pre-processed to filter out the unnecessary features and the pre-processing is also done by the WEKA tool. After the pre-processing is done, the data is fed into the different classifiers of machine learning to fetch the evaluation result.

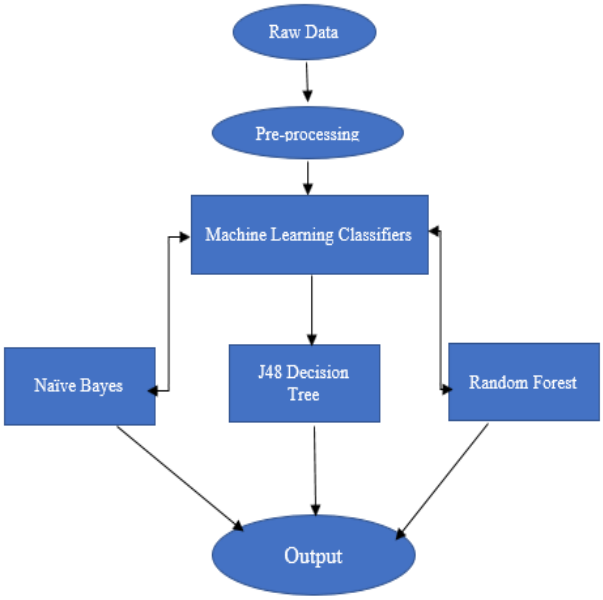

Figure 3: Flow of the experiment

A. Evaluation Metric

There are certain performance metrics which are used and are as follow:

- **TPR:** TPR is defined as the ratio classified apps that are correct which contains malware to the number of malicious apps in totality.
  TPR= TP / (TP + FN)

Where TP: True positives (correctly identified malicious apps), and FN: False negatives which are misclassified malware instances.

- **FPR:** This is given as the ratio wrongly classified benign apps to the number of benign apps in totality.
  FPR= FP / (TN + FP)

Where FP: false positives and TN: true negatives.

- **Precision:** This rate is positive predictive and is stated as below:
  Precision= FP / (TP + FP)
- The time taken to test the models are given in seconds. These models are tested on 64 bit, Windows 10 PC which is having 12 GB of RAM.

B. WEKA Tool

The tool which we have used here to analyze the dataset is the WEKA Tool. It is a software that is open-source that pre-process the data first according to the need for an experiment, apply several machine learning algorithms, and create a visual representation. We take raw data as input which may have several null values and unwanted attributes, the pre-processing phase of WEKA helps to clean all that. Next, depending on the kind of model which you need to develop you may have to select from the given options like Cluster, Classify, or Associate. Under each selection you have several machine learning algorithms, you may select an algorithm of your choice and the particular dataset to get the results. Also, the same dataset can be applied to different models, and then the output can be compared to check which model gives the best output to meet your purpose. WEKA is open-source under GNU public licensing and is considered platform-independent as the code is written in java and it provides the user with a graphical user interface to interact with files and provides visual graphs and curves for analysis [19].

C. Classifiers

1. ***Naïve Bayes*:** This is a classifier of machine learning, which comes under the group of supervised learning and is based on probabilistic logic. This algorithm assumes that all the values for particular features are not dependent on any of the other feature's value. In this classification, we try to find out the best hypothesis (h) for the give data (d). To find out the best hypothesis the easiest solution is to use our prior knowledge. The theorem provides us with a method of calculating the best hypothesis provided the knowledge previously gained. The theorem here says:
   P(hy|da) = (P(da|hy) * P(hy)) / P(da)
   Where P(hy|da) here is the probability of the given hypothesis in which the data (da) and is known as the posterior probability. P(da|hy) here the probability of the data da to provided hypothesis hy. P(hy) here is the probability of the true hypothesis and is known as prior knowledge. P(da) here is the probability of data provided [20].
2. ***J48:*** J48 algorithm is a classifier that belongs to the decision tree group which is in turn part of a supervised learning approach in which the data input is continuously split according to particular parameters. The tree can be categorized into two types of nodes such as decision nodes and leaves. The additional features of J48 are decision tree pruning which means to prune or not to traverse the next node if we find the best solution, derivation of rules, accounting for missing values. The basic steps of this algorithm are:
   - If the instance belongs to the same class, it is denoted by leaf and then the leaf is returned by labeling the same class.

- The attributes are tested to calculate the potential information and then the information gain is calculated.
- Then the best solution is selected for branching [21].

3. ***Random Forest:*** Random Forest is a classifier of machine learning in which a supervised learning approach for classification is. We know that the forest terminology is used for a collection of trees, more trees mean more robust forest. This algorithm creates trees randomly on the dataset and selects the best suited by voting. Due to this approach, it eliminates the issue of over-fitting by averaging the values. These votes can be weighted or un-weighted.[22] The steps followed by the algorithm are:
   - Bootstrapping of a dataset is done to eliminate the data which is not required.
   - A tree is created using a random number of attributes. The attributes form the leaves and nodes using the tree building algorithm.
   - The trees are not pruned and are allowed to grow to its fullest.

VI. RESULTS AND DISCUSSIONS

This section contains the result of our experiment which was carried out on two different datasets and the classifiers applied were three namely: J48 Decision Tree, Naïve Bayes, and Random Forest. The tool was utilized as an open-source tool, WEKA. The datasets were optimized using this tool only. For all the classifiers the value of percentage split was set to 70% and 30%. Splitting the dataset means divided it into two parts: one for the testing and the other for training. The same configuration for both the datasets was used in order to maintain the consistency. The given table 2 and table 3 clearly defines the result of each classifier used in accordance with different parameters.

Figure 4 shows the classification of classes resulted which are class S (Malware) and class B (Benign), and Figure 5 shows some of the attributes of the Malgenome dataset.

TABLE II. RESULTS ON MALGENOME DATASET

Classifier	TPR	FPR	Precision	F-Measure
Naïve Bayes	0.961	0.038	0.962	0.962
J48	0.989	0.015	0.989	0.989
Random Forest	0.991	0.014	0.991	0.991

TABLE III. TABLE 3: RESULTS ON DREBIN DATASET

Classifier	TPR	FPR	Precision	F-Measure
Naïve Bayes	0.835	0.126	0.862	0.837
J48	0.972	0.033	0.972	0.972
Random Forest	0.984	0.022	0.984	0.984

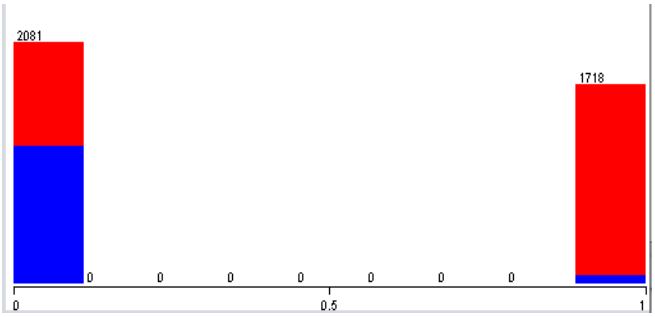

Figure 4 : Shows the Malgenome dataset results

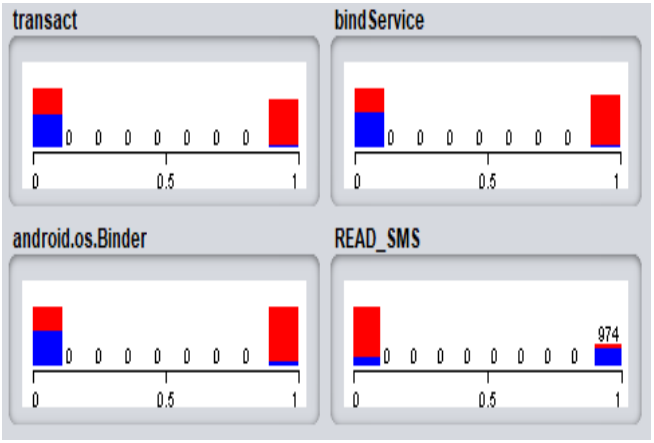

Figure 5: Some of the attributes visualized of Malgenome

Figure 6 shows the classification of classes resulted which are class S (Malware) and class B (Benign), and Figure 7 shows some of the attributes of the Drebin dataset.

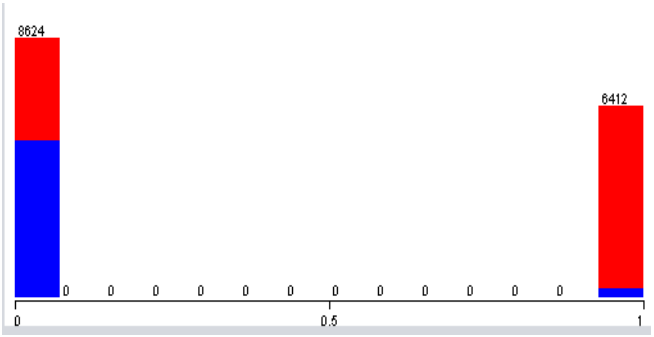

Figure 6 : Shows the Drebin dataset results

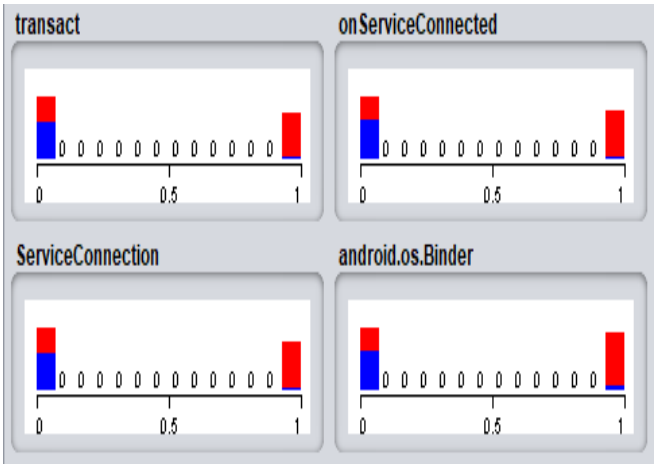

Figure 7: Some of the attributes visualized of Drebin

VII. CONCLUSION

Machine learning is a branch of computer science which suggests that if we input correct data to the computer then it can learn and perform future actions with the help of that training data and its experience. The analysis done here is how the different machine learning classifiers work for a given particular dataset. The different malware datasets used were Malgenome and Drebin datasets. We tried to analyze and summarize the accuracy of three classifiers of machine learning which are J48 Decision Tree, Naïve Bayes, and Random Forest on these datasets. The tool used for experimentation is the open-source tool WEKA. The results of each are depicted by the tables.

REFERENCES

1. J. Sahs and L. Khan, "A machine learning approach to android malware detection," *Proc. - 2012 Eur. Intell. Secur. Informatics Conf. EISIC 2012*, pp. 141–147, 2012.
2. J. Li, L. Sun, Q. Yan, Z. Li, W. Srisa-An, and H. Ye, "Significant Permission Identification for Machine-Learning-Based Android Malware Detection," *IEEE Trans. Ind. Informatics*, vol. 14, no. 7, pp. 3216–3225, 2018.
3. J. Qiu, W. Luo, L. Pan, Y. Tai, J. Zhang, and Y. Xiang, "Predicting the Impact of Android Malicious Samples via Machine Learning," *IEEE Access*, vol. 7, pp. 66304–66316, 2019.
4. Y. Zhou and X. Jiang, "Dissecting Android malware: Characterization and evolution," *Proc. - IEEE Symp. Secur. Priv.*, no. 4, pp. 95–109, 2012.
5. F. Musumeci *et al.*, "An Overview on Application of Machine Learning Techniques in Optical Networks," *IEEE Commun. Surv. Tutorials*, vol. 21, no. 2, pp. 1383–1408, 2019.
6. A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash, "Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications," *IEEE Commun. Surv. Tutorials*, vol. 17, no. 4, pp. 2347–2376, 2015.
7. H. S. Ham and M. J. Choi, "Analysis of Android malware detection performance using machine learning classifiers," *Int. Conf. ICT Converg.*, pp. 490–495, 2013.
8. B. Amos, H. Turner, and J. White, "Applying machine learning classifiers to dynamic android malware detection at scale," *2013 9th Int. Wirel. Commun. Mob. Comput. Conf. IWCMC 2013*, pp. 1666–1671, 2013.
9. A. Shabtai, U. Kanonov, Y. Elovici, C. Glezer, and Y. Weiss, "'Andromaly': A behavioral malware detection framework for android devices," *J. Intell. Inf. Syst.*, vol. 38, no. 1, pp. 161–190, 2012.
10. R. Bost, R. A. Popa, S. Tu, and S. Goldwasser, "Machine Learning Classification over Encrypted Data," no. February, pp. 8–11, 2015.
11. M. Ghorbanzadeh, Y. Chen, Z. Ma, T. C. Clancy, and R. McGwier, "A neural network approach to category validation of Android applications," *2013 Int. Conf. Comput. Netw. Commun. ICNC 2013*, pp. 740–744, 2013.
12. S. Y. Yerima, S. Sezer, and I. Muttik, "High accuracy android malware detection using ensemble learning," *IET Inf. Secur.*, vol. 9, no. 6, pp. 313–320, 2015.
13. A. A. A. Samra, K. Yim, and O. A. Ghanem, "Analysis of clustering technique in android malware detection," *Proc. - 7th Int. Conf. Innov. Mob. Internet Serv. Ubiquitous Comput. IMIS 2013*, pp. 729–733, 2013.
14. T. Chen, Q. Mao, Y. Yang, M. Lv, and J. Zhu, "TinyDroid: A lightweight and efficient model for android malware detection and classification," *Mob. Inf. Syst.*, vol. 2018, 2018.
15. L. Yu, Z. Pan, J. Liu, and Y. Shen, "Android malware detection technology based on improved Bayesian classification," *Proc. - 3rd Int. Conf. Instrum. Meas. Comput. Commun. Control. IMCCC 2013*, pp. 1338–1341, 2013.
16. L. Tenenboim-Chekina *et al.*, "Detecting application update attack on mobile devices through network featur," pp. 91–92, 2014.
17. H. H. Kim and M. J. Choi, "Linux kernel-based feature selection for Android malware detection," *APNOMS 2014 - 16th Asia-Pacific Netw. Oper. Manag. Symp.*, 2014.
18. *et al.*, "Analysis of Android Vulnerabilities and Modern Exploitation Techniques," *ICTACT J. Commun. Technol.*, vol. 05, no. 01, pp. 863–867, 2014.
19. W. Ahmed, A. Saeed, A. Salah, and E. Abdala, "A Comparative Study for Machine Learning Tools Using WEKA and Rapid Miner with Classifier Algorithms Random Tree and Random Forest for Network Intrusion Detection," vol. 4, no. 4, pp. 749–752, 2019.
20. E. P. F. Lee *et al.*, "An ab initio study of RbO, CsO and FrO (X2∑+; A2∏) and their cations (X3∑-; A3∏)," *Phys. Chem. Chem. Phys.*, vol. 3, no. 22, pp. 4863–4869, 2001.
21. G. Kaur, "Improved J48 Classification Algorithm for the Prediction of Diabetes," vol. 98, no. 22, pp. 13–17, 2014.
22. F. Livingston, "Implementation of Breiman's Random Forest Machine Learning Algorithm," *Mach. Learn. J. Pap.*, pp. 1–13, 2005.