**Title: Android Malware Detection using Machine Learning**

**Authors: Indu Chawla, Dhananjay Singh, Shubham Karpa**

**Abstract:**

Being the most used operating system makes Android vulnerable to various kinds of malware attacks. This is due to the fact that most applications require several permissions which are necessary for installation, or smooth operability. Some android apps are not certified by legitimate organizations and may contain malware which can steal private user information. This has increased the interest of applying machine learning algorithms for android malware detection. In this research paper we evaluate Naive Bayes, Decision Tree, Random Forest, and Support Vector Machine which are algorithms using machine learning which are popularly used for detecting malicious android applications based on permissions model. The results show that RF performs the best giving an accuracy of   
98.15% using k-fold cross-validation for k=5 and a mean accuracy of 97.79%.

Index Terms—Android, Malware, Random Forest, Support Vector Machine, Machine Learning.  
**Introduction:**

Being one of the most popular operating systems, through the years it has become easier for malware developers to be able to develop and make popular Android applications which are able to not only affect the functioning of devices in which they are installed, but also be able to steal confidential data.  
This has led to theft of personal data and enabled and popularized cyber crime.

Android has a permissions based mechanism that applies restriction on an application for security purposes. A number of permissions are required at the time of installation and startup of an android application such as the ability to access the users contact list, internet, photo gallery etc. These permissions give the applications ability to access these resources, and this can only be revoked by uninstalling the application. Certain applications sometimes request for permissions that they do not need. For example these applications request access for user’s phone contacts or photo gallery and hence causes a breach of private data. Examples of such malicious apps include some photo editor apps which are developed and released into the playstore. Later on when the users download the app which performs like any other benign app but it also accesses the user's private photo gallery and breaches privacy. It’s not necessary for an application to be a virus – the case can also be so that the application has access to privileges it does not need. Some applications gain internet access and use GPS to gain information about the user's location. Hardware features such as touchscreen, Bluetooth and camera can also be accessed by certain malicious applications. Malware applications also use a lot of internal memory of the device. These applications also lead to a greater battery consumption and in turn damage the device by overheating the battery.

Android smartphones are becoming increasingly popular. It is assessed that there will be roughly 6.1 billion smartphone clients by 2020 [36, 37]. On the other side, Android devices are an increasingly attractive target for online criminals who try to hold personal details (i.e., location, contact numbers, accounts, photos, etc.) [[3](https://www.mdpi.com/2076-3417/9/2/277/htm#B3-applsci-09-00277)]. In addition to this, most of the Android devices do not use anti-virus or malware detection applications [38, 39, 40]. The number of malware attacks is increasing at a high rate. According to a report by Kaspersky Lab, mobile users in 2018 faced one of the strongest criminal attacks. New records were set in terms of both number of mobile banking Trojans detected and number of attacked users. This has greatly increased the requirement for an effective algorithm to be able to detect malware and prevent such attacks.

Researchers have proposed two approaches to be able to detect malware, specifically for Android: static analysis and dynamic analysis.

Static Analysis approach comprises of three kinds of analysis;

1. The Signature-based strategy uses specific features and signs for the identification of specific malware. This is also a drawback, because it will not be able to detect previously unidentified malware.
2. The Permission-based strategy uses specific permissions to be able to build onto a classification of malware.
3. The Component-based techniques entirely decompiles the APP and draws and inspects the definition and byte code connections of certain important components such as activities and services. These are further used to identify exposure.

This analysis includes features that are collected without execution of code. Drawback of static analysis is that it fails to deal with the part of code which is downloaded during execution.

Lightweight on-device detection using permissions, network access, API calls, etc. using SVM was first put to floor by Around et al [4]. Yerima et al proposed a random forest ensemble learning model which used API calls, permission, intent and the commands which are embedded. [20],[21] Varsha et al [22] used a method which investigates random forest, rotation forest and SVM on 3 datasets. Additionally, Cen et al [25] proposed a method which was based on using API calls by decompiling the code and the permissions it uses. The application of a discriminative robust model based on regularized logistic regression was done. RLR is then subsequently compared to the SVM, Decision Tree, k-NN, and similar models.

In dynamic analysis the features of an android application include CPU and battery consumption, number of processes running and number of short messages, etc. When a malware application is on our device then the battery consumption also increases. Therefore, the battery gets heated up and eventually damages it. Greater CPU consumption will also slow our android mobile device. We can use this approach to find differences in the dynamic features of benign and malicious applications and hence detect malware. Dynamic analysis can execute the actual code which is executed by the app. The primary benefit of this strategy is that it will detect dynamic code loading and will be able to record the behavior of the application while it is running. This technique, however, will not be able to indicate the amount of code that will get executed when the application runs. It might be possible that the application fails to execute the malicious code when the features are being recorded. Another drawback of this technique is the difficulty in executing it as compared to static analysis because of the overhead during application execution.

Mahindru and Singh [29] applied to a lot of machine learning techniques, which were individual in nature, which included Naive Bayes, Decision Tree, Random Forest, Simple Logistic. They did this by extracting 123 dynamic permissions from over 11000 applications. The performance of Simple Logistic was better of all, however the accuracy with which the malware was classified was similar in Random Forest, Simple Logistic and Decision tree.

AntiMalDroid[26] created a framework to be able to detect malware which was based on dynamic analysis behaviour. This method uses logged behaviour sequence as features with SVM, and DroidDolphin [27], was also able to use SVM with features obtained in a dynamic manner. Afonso et al [28] used dynamic API calls and system call traces. He applied SVM, J48, IBk, BayesNet K2, BayesNet TAN, Random Forest and Naive Bayes.

In this research work we have used the static analysis for the classification of android applications. The static features that are considered in the paper are permissions that are extracted from the android manifest file. The following paper provides with an implementation for the classification problem by employing several algorithms which use machine learning such as support vector machine, random forest decision tree, and Naïve Bayes. We have used permissions of 3799 android applications. This has been to support the effectiveness of our work. Various performance metrics have been evaluated for finding out the overall accuracy of our classifiers.

**Related Work:**

As discussed in the introduction, the two major methods of detecting malware are static analysis and dynamic analysis.

Static analysis will decompile the APK file and will extract the major features of the application. Crowdroid [5] is a machine-learning-based framework that detects malware in an android device. Crowdroid will basically analyze the total times that a system call has been issued whenever there is execution of any action that uses user interaction. Benign and malware applications issue different types and number of system calls.

Droidmat detects malware by using manifest and API calls [8]. First, the extraction of information is done from the manifest, followed by examination of data by reviewing the API calls that were made by the system. K-means algorithm is applied to do an analysis of the system.

Lightweight on-device detection using permissions, network access, API calls, etc. using SVM was first put to floor by Around et al [4]. Yerima et al proposed a random forest ensemble learning model which used API calls, permission, intent and the commands which are embedded. [20],[21] Varsha et al [22] used a method which investigates random forest, rotation forest and SVM on 3 datasets which uses a method that extracts static features from the files. Sharma and Dash used the API calls and it’s permissions to create detective systems that are based on Naïve Bayes and k-NN.

Detection of malware piggybacked onto benign apps was done in DAPASA [24]. The features are fed into machine learning algorithms. These are Random Forest, Decision Tree, k- NN and PART. Out of these, Random Forest is the most accurate. Additionally, Cen et al [25] proposed a method which was based on using API calls by decompiling the code and the permissions it uses. The application of a discriminative robust model based on regularized logistic regression was done. RLR is then subsequently compared to the SVM, Decision Tree, k-NN, and similar models.

The features, intents and permissions, were used to train machine learning models and classifier fusion. This method used for improving performance was performed by Idrees et al [31]. He compared the performance of MLP, Decision Tree, Decision Table, Random Forest, Sequential Minimal Optimization and Naive Bayes. Three classifiers DT, MLP and decision table were combined to using three schemes: average of probabilities, product of probabilities and majority voting.

Ni et al [32] proposed a real-time behaviour system. The system uses real time features like user operations, it records API calls and permission uses. TaintDroid [9] tracks multiple data sources in a dynamic manner. The weakness of TaintDroid is that manual efforts to traverse user interfaces are needed to cover dangerous functionality effectively. DREBIN [11] creates a combination of both machine learning and static analysis. It is able to get high detection accuracy because it uses as many features as possible for detecting malware. The drawback with this is that the more features you use, the higher is the overhead.

Mahindru and Singh [29] applied to a lot of machine learning techniques, which were individual in nature, which included Naive Bayes, Decision Tree, Random Forest, Simple Logistic. They did this by extracting 123 dynamic permissions from 11000 applications. The performance of Simple Logistic was better of all, however the accuracy with which the malware was classified was similar in Random Forest, Simple Logistic and Decision tree.

AntiMalDroid[26] created a framework to be able to detect malware which was based on dynamic analysis behaviour. This method uses logged behaviour sequence as features with SVM, and DroidDolphin [27], was also able to use SVM with features obtained in a dynamic manner. Afonso et al [28] used dynamic API calls and system call traces. He applied SVM, J48, Ibk, BayesNet K2, BayesNet TAN, Random Forest and Naive Bayes.

Dynamic analysis approaches require input which is effective enough to complement the path of execution.

Petra’s et al. [10] has been able to demonstrate analysing the program in a dynamic manner is not sufficient to assure Android security. Therefore, data mining and machine learning techniques are being researched in order to use permissions to be able to detect Android malware.

**Proposed Approach:**

In our research we focus on an approach for Android Malware Detection.

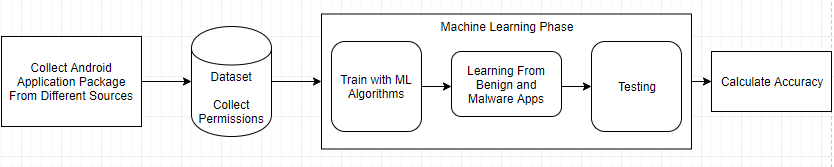
Our dataset comprises of 3799 android application samples and 216 attributes. This includes permissions and API calls of the applications. We have selected 1260 malware apps from the Android Malgenome project and 2539 benign apps. It consists of information that has been extracted from four datasets.

Two of the datasets used, namely Android Malgenome project [3] and DREBIN [11] were available in public domain. We are grateful to the respective authors and the opportunity to be able to use these datasets.

The dataset is divided into two parts. 70% of it is training data and 30% is the test data. We use the testing dataset to create models of various machine learning algorithms. These are the algorithms on which we will do static analysis.

The research work uses classification algorithms like Naive Bayes, Decision Tree, Random Forest and Support Vector Machine to create the models for respective machine learning algorithms. The machine learning phase uses 10-fold cross validation for Random forest ensemble learning algorithm which outputs the highest accuracy for its model.

This paper proposes Static analysis which uses the features that have been extracted from the APK manifest files. The training dataset trains the classifiers on which the testing data is then used to predict the accuracy of the models created. The accuracy of different models have been compared in the results sections. The below flowchart show the brief description of the proposed work flow in this research.



**Implementation:**

Android applications are coded in Java programming language. Then they are collected together and bundled in an Android Package, commonly abbreviated as APK. These applications keep running in a different process. There is an XML descriptor document called AndroidManifest.xml of which these applications are made up of. Information required by the Android framework of the app is contained in this document. The document determines the applications permissions, packages, APIs and libraries. There are two different android malware detection technique based on features employed to classify an application: static and dynamic approaches.

This manifest file also contains all the information about the android app along with system permissions under the <uses-permissions> tag.

Some malware tries to gather user private information using these permissions which could be as follows:

1. ACCESS\_FINE\_LOCATION: Gives the app user's precise location.
2. CALL\_PHONE: App can call without having to go into the systems dial user-interface.
3. READ\_CONTACTS: Access all of user's contacts.
4. CAMERA: Allows app to access all the camera features.
5. SEND\_SMS: Gives permission to app to send SMS.
6. WRITE\_CALENDAR: Gives permission to write on the user's calendar.

Some of these permissions can lead to a breach of private data. . In [41], approving permissions "READ\_EXTERNAL\_STORAGE" and "WRITE\_MEDIA\_STORAGE" will give an app access to read the SD card which usually stores most of a user’s personal data.

DroidSwan [42] lists down a total of 8 permissions, which give an application excessive privilege. These include "WRITE\_APN\_ SETTINGS", "INTERNET", "WRITE\_SMS" and "WRITE\_EXTERNAL\_STORAGE".

The study in [43] demonstrates that malware generally utilizes certain permissions not usually requested by benign Apps. For example, the permission "sendTextMessage” will be able to use the text message service without showing any notification. The study has also researched statistics on both malicious and benign Apps. The study has thus been able to recognize a difference between 20 most requested permissions in the two data-sets.

The dataset which comprised of both permissions and API calls was split into training and test data. After which machine learning phase was implemented.

We’ve applied Naive Bayes, Decision Tree, Support Vector Machine and Random Forest in this report. We performed k-fold cross-validation on random forest. This has proved to be the most efficient method, where in for the fold value k = 10 we have obtained the accuracy to be 97.79%.

**Theory:**

**Naive Bayes:** An assumption is made regarding the probability that a given attribute belongs to a specific class will be independent of all other attributes. We use the probability of a class value, knowing the value of a specific attribute. It is known as conditional probability. To get to the probability of a data instance, we can simply multiply the conditional probability of each of the attributes. We then take the probability of a data instance of every class. We select the class value with the greatest probability, and make a prediction. The calculations are greatly simplified due to the assumption of independence of an attribute of a specific class from all the other attributes, which is a very big assumption. Regardless, this is an extremely quick and effective algorithm.

**Decision Tree:** A decision tree is in the form of a tree, with one node being the tree, and with any node which has an outgoing chain being the test node. All the values are in a line. All the points of splitting are used in this numerical procedure, and the Gini cost function is applied on them, in order to test them. The function helps determine the purity of the nodes. This is measured by the amount of mixture of the data assigned for training that is assigned to a specific node. We perform splitting, and that is done till the nodes have either of the following; either a minimum number of training examples, or the depth of the tree reached a specific number. Size of tree is 83 and number leaf nodes are 42.

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

**Random Forest**: In this method, an independent random vector is sampled. Values dependent on this vector generate several tree predictors. This further goes for all the trees in the forest. Further, the generalization error is calculated of the forest having classifiers. A factor of this calculation is the strength of the inner trees and their correlation. After that each node is split using random selection of features which will further yield error rates compared to Adaboost, it is favorable and robust.

Steps:

1. A number smaller than the variables is taken consisting of ntree, mtry and number of trees to grow.
2. For i=1 for ntree:
3. We draw a sample of data. “Out-of-bag” data is the data that was not there in this bootstrap sample.
4. The best split is chosen among mtry randomly selected variables for growing a “random” tree. The tree is not pruned back and grown to its maximum size.
5. Tree is used to predict the data that was not there in the sample.
6. We will use the predictions with maximum votes on the out-of-bag data in the end.
7. We will be able to predict the test data by using the maximum votes in the tree ensemble.

Split stopping rule: We implement some large tree procedures in order to trim the tree in an upward manner.

Class assignment: We assign most of the class in the node in a normal manner. This is done unless we already have available a strong prior probability of the class.

**Support Vector Machine**: The main objective is to find the hyperplane in the n-dimensional space that classifies the data point. n in this case is the total number of features. All the data items are plotted as a point in this space. The numerical value of the coordinate will be the value of n. Afterwards, we perform classification. This is done by finding a hyper-plane. This plane should be able to differentiate both the classes. Many hyperplanes are possible and so the objective is to find the best possible hyperplane with the maximum distance between data points of both classes. The dimension of the hyperplane will depend on the number of features. A hyperplane is a subspace with dimension one less than that of its ambient space. For example: for a given space of three dimensions a hyperplane would be two dimensional. Support vectors are points closest to the hyperplane.

Once the machine-learning phase is completed the evaluation of the training models is done using various metrics such as TPR, FPR, F-measure, precision and recall. Confusion matric is also built on the basis of the performance of the four classifiers. Mean accuracy of each of the classifiers is also calculated and displayed in the results section.

**Results:**

In this paper four machine learning algorithms were applied for effective Android Malware Detection which are Naive Bayes, Support Vector Machine, Decision Tree and Random Forest. The algorithms were tested against a collection of 3799 samples having 216 attributes. We split the dataset into two parts, 70% of which was for training and 30% of which was for test. The best algorithm was random forest with 10-fold cross-validation. The average accuracy of the same was 97.78 percent.

Once the training model was developed from whole dataset we used four machine-learning algorithms to model and evaluate the dataset. A confusion matrix was built from the response of our classifiers.

**Sample Confusion Matrix:**

|  |  |  |
| --- | --- | --- |
|  | **T** | **F** |
| **T** | TP | TN |
| **F** | FP | FN |

Below given four metrics define the confusion matrix:

**True Positives (TP)**: Benign applications that are correctly identified

**True Negatives (TN)**: Malicious applications that are correctly identified

**False Positives (FP)**: Malware applications which were incorrectly identified

**False Negatives (FN)**: Benign applications which were incorrectly identified

Given below are performance/evaluation metrics:

**True positive rate (TPR):** % of the benign applications which were identified correctly.

TP/ (TP+FN)

**False positive rate (FPR):** % of malware applications which were identified incorrectly.

FP/ (TN+FP)

**Precision (TP) :** Which had a prediction of Yes

**F-score:** Weighted average of TPR and precision.

Given below are confusion matrix for all 4 classifiers.

**Table I:** **Normalized Confusion Matrix for Decision Tree**

|  |  |  |
| --- | --- | --- |
| Actual S | 0.98 | 0.02 |
| Actual B | 0.03 | 0.97 |
|  | Predicted S | Predicted B |

**Table II: Normalized Confusion Matrix for Naive Bayes**

|  |  |  |
| --- | --- | --- |
| Actual S | 0.94 | 0.06 |
| Actual B | 0.35 | 0.65 |
|  | Predicted S | Predicted B |

**Table III:** **Normalized Confusion Matrix for SVM**

|  |  |  |
| --- | --- | --- |
| Actual S | 0.99 | 0.01 |
| Actual B | 0.06 | 0.94 |
|  | Predicted S | Predicted B |

**Table IV:** **Normalized Confusion Matrix for Random Forest**

|  |  |  |
| --- | --- | --- |
| Actual S | 1 | 0 |
| Actual B | 0.02 | 0.98 |
|  | Predicted S | Predicted B |

* 1. The Naive Bayes has the lowest TP rate and lowest accuracy out of all the other 5 classifiers.
  2. Random Forest is the most accurate.
  3. DT and SVM have a better performance than the Naive Bayes classifier.

Android Malware can be detected by a number of classifier machine learning algorithm but we find out that Random forest with 10 fold cross validation gave the highest accuracy. In a nutshell we can conclude that Random Forest is capable of detecting almost all of malware present in the dataset.

The mean accuracy of all the classifiers are shown in below:

**TABLE V: RESULTS OF PERFORMANCE**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Algorithms** | **Performance Metrics** | | | | | |  |
| TP rate | FP rate | Precision | Recall | F-measure | ROC area | Accuracy (%) |
| Naive Bayes | 0.95 | 0.05 | 0.95 | 0.95 | 0.95 | 0.99 | 81.67 |
| Decision Tree | 0.98 | 0.02 | 0.98 | 0.98 | 0.98 | 0.98 | 96.49 |
| Random Forest | 0.99 | 0.01 | 0.99 | 0.99 | 0.99 | 1 | 97.79 |
| SVM | 0.99 | 0.01 | 0.99 | 0.99 | 0.99 | 0.99 | 97.6 |

Naive Bayes algorithm outputs accuracy of 81.67% as shown in table I. This classifier gave a low accuracy which could have been improved by using weighted Naive Bayes algorithm. This way more relevant features will be included in the new permission set and the algorithm will perform better. Irrelevant or redundant features will be deleted and only a few number of relevant features will be used to provide better accuracy.

Support Vector Machine algorithm outputs accuracy of 97.60%. SVM classifier was used because of the optimal marginal gap between the separating hyper planes and this gives better results with the test dataset. SVM works better in two class classification and when number of features are less.

Decision Tree classifier had the min\_samples\_leaf is set to 5 and min\_samples\_split is set to 2.This classifier outputs accuracy of 97.49%. This algorithm was chosen because the risk of overfitting is less than in SVM. The size of the decision tree was 83 with 42 leaves. This implies that the decision tree is small. It has at the most 42 decision branches. Pruning algorithms can be used to prune the tree size and solve the problem of overfitting.

Random Forest algorithm with 10-fold cross validation outputs the highest accuracy of 97.79%. We have only one parameter, i.e. k. k is the total number of groups into which a sample of data will be split. Thus, the algorithm gave better results for k=10 than for k=5. This classifier was chosen because it works best when the number of trees are increased since the model will not over fit. This classifier also handles the missing values and maintains the overall accuracy of the dataset. Random forest worked better than SVM because of large number of features in the dataset.

The classifier was trained by performing experiments on both training and test samples. This was done to assess the functioning of the proposed method. The experiments conducted bring to light a simpler view of predicting the functioning of different types of machine learning classifiers.

**Conclusion:**

There has been a huge evolution of mobile device technology. The popularity of Android has enabled the spread of Android Malware. This has resulted in an immediate need for an operative android malware detection system. This system should not only be able to detect existing malware, but also unidentified, zero-day malware.

The work has been implemented by a framework of machine learning algorithms. Four efficient algorithms namely, Support Vector Machine, Decision Tree, Naive Bayes and Random Forest with k-fold cross-validation for classifying Android Applications as malware or benign are used. Several android permission are used in the model to categorize it into one of the two mentioned categories. Along with the permissions API calls have also been used in this paper. The most accurate algorithm was the Random forest with 10-fold cross-validation. It’s average accuracy was calculated to be 97.78%.

Regarding future work, we will train models with larger dataset. This will give us more precise results. Other static features like API calls will be included. java.lang and android.telephony, are the most frequently used API in benign and malware applications, and factors like this can be considered in future designs.

Dynamic analysis which includes features like CPU consumption or battery consumption will also be included in the dataset for better results. Malware applications can also be categorized into their respective families for a more specific malware detection (Trojan, Infosteal, etc.).

**References:**

[1] Suleiman Y. Yerima and Sakir Sezer, “DroidFusion : A Novel Multilevel Classifier Fusion

Approach for Android Malware Detection” in IEEE Transactions on Cybernetics January 2018

[2]<http://www.statista.com/statistics/271774/share-ofandroid-platforms-on-mobile-devices-with>-android-os/

[3]Yajin Zhou, Xuxian Jiang, "Dissecting Android Malware: Characterization and Evolution," Proceedings of the 33rd IEEE Symposium on Security and Privacy (Oakland 2012), San Francisco, CA, May 2012

[4] Symantec, "latest intelligence for March 2016," in Symantec social Blog, 2016.

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

[6] Y. Zhou, Z. Wang, W. Zhou, and X. Jiang, Hey, you, get off of my market: Detecting malicious apps in official and alternative Android markets. In Proceedings of the 19th Annual Network & Distributed System Security Symposium, Feb. 2012.

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

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

[9] W. Enck, P. Gilbert, S. Han, V. Tendulkar, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth, “Taintdroid: an information-flow tracking system for real time privacy monitoring on smartphones,” ACM Transactions on Computer Systems (TOCS), vol. 32, no. 2, p. 5, 2014.

[10] T. Petsas, G. Voyatzis, E. Athanasopoulos, M. Polychronakis, and S. Ioannidis, “Rage against the virtual machine: hindering dynamic analysis of android malware,” in Proceedings of the Seventh European Workshop on System Security. ACM, 2014, p. 5.

[11] D. Arp, M. Spreitzenbarth, M. Hübner, H. Gascon, K. Rieck, and C. Siemens, “Drebin: Effective and explainable detection of android malware in your pocket,” in Proceedings of the Annual Symposium on Network and Distributed System Security (NDSS), 2014

[12] H.A. Alatwi. Android malware detection using category-based machine learning classifiers. Masters

Thesis, June 2016.

[13]J. Zhang, S. Saha, G. Gu, S.-J. Lee, and M. Mellia, “Systematic mining of associated server herds for malware campaign discovery,” in Distributed Computing Systems (ICDCS), 2015 IEEE 35th International Conference on. IEEE, 2015, pp. 630–641.

[14] C.-Y. Huang, Y.-T. Tsai, and C.-H. Hsu, “Performance evaluation on permission based detection for android malware,” in Advances in Intelligent Systems and Applications-Volume 2. Springer, 2013, pp. 111–120.

[15] Pan, Y. Zhongyang, Z. Xin, B. Mao, H. Huang, Defensor: Lightweight and Efficient Security-Enhanced Framework for Android, in Trust, Security and Privacy in Computing and Communications (TrustCom), 2014 IEEE 13th International Conference in 2014, pp. 260–267. IEEE

[16] G. Canfora, E. Medvet, F. Mercaldo, and C. A. Visaggio, “Detecting android malware using sequences of system calls,” in Proceedings of the 3rd International Workshop on Software Development Lifecycle for Mobile, 2015, pp. 13–20.

[17] M. Dimjašević, S. Atzeni, I. Ugrina, and Z. Rakamaric, “Evaluation of Android Malware Detection Based on System Calls,” in Proceedings of the 2016 ACM on International Workshop on Security And Privacy Analytics, 2016, pp. 1–8.

[18] Li, Wenjia, Jigang Ge, and Guqian Dai. ”Detecting malware for android platform: A svm-based approach.” Cyber Security and Cloud Computing (CSCloud), 2015 IEEE 2nd International Conference on. IEEE, 2015.

[19] Atici, Mehmet Ali, Seref Sagiroglu, and Ibrahim Alper Dogru. ”Android malware analysis approach based on control flow graphs and machine learning algorithms.” Digital Forensic and Security (ISDFS), 2016 4th International Symposium on. IEEE, 2016.

[20]S. Y. Yerima, S. Sezer, and I. Muttik. ”Android malware detection: An eigenspace analysis approach” In proc. Science and Information Conference (SAI 2015), London, UK, 28-30 July 2015, pp.1236-1242.

[21]S. Y. Yerima, S. Sezer, and I. Muttik. High accuracy Android malware detection using ensemble learning. IET Information Security, Vol 9, issue 6, 2015, pp. 313-320.

[22]M. Varsha, P. Vinod, & K. Dhanya. Identification of malicious Android app using manifest and opcode features. Journal of Computer Virology and Hacking Techniques, 2016, pp. 1-14.

[23]A. Sharma and S. Dash, “Mining API calls and permissions for Android malware detection” in Cryptology and Network Security. Springer International Publishing, 2014, pp. 191205.

[24]M. Fan, J. Liu, W. Wang, H. Li, Z. Tian and T. Liu. DAPASA: Detecting Android Piggybacked Apps through Sensitive Subgraph Analysis. IEEE Transactions on Information Forensics and Security, Vol. 12, Issue 8, March 2016, pp. 1772-1785.

[25]L. Cen, C. S. Gates, L. Si, and N. Li. A Probabilistic Discriminative Model for Android Malware Detection with Decompiled Source code. IEEE Transactions on Secure and Dependable Computing, Vol. 12, No. 4, July/August 2015.

[26]M. Zhao, F. Ge, T. Zhang, and Z. Yuan, ”Antimaldroid : An efficient SVM based malware detection framework for android” In C. Liu, J. Chang, and A. Yang, editors, ICICA (1), volume 243 of Communications in Computer and Information Science, Springer, 2011. pp. 158166.

[27]W.-C. Wu and S.-H. Hung,”Droiddolphin: A dynamic Android malware detection framework using big data and machine learning” In proc. 2014 ACM conf. on Research in Adaptive and Convergent Systems, (RACS ’14), NY, USA, pp. 247-252.

[28]V. M. Afonso, M. F. de Amorim, A. R. A. Gregio, G. B. Junquera, and P. L. de Geus. Identifying Android malware using dynamically obtained features. Journal of Computer Virology and Hacking Techniques, 2014.

[29]A. Mahindru and P. Singh ”Dynamic Permissions based Android malware detection using machine learning techniques” In proc. 10th Innovations in Software Engineering Conference, Jaipur, India, Feb. 5-7, 2017. pp 202-210.

[30]M.-Y. Su, J.-Y. Chang, and K.-T. Fung ”Machine Learning on Merging Static and Dynamic Features to identify malicious mobile apps” In proc. 9th Int. Conf. on Ubiquitous and Future Networks (ICUFN), 2017, Milan, Italy, 4-7 July 2017. pp. 863-867.

[31]F. Idrees, M. Rajarajan, M. Conti, T. M. Chen, Y. Rahulamathavan. PIndroid: A novel Android malware detection system using ensemble learning methods. Computers & Security, Vol 68, July 2017, pp. 36-46.

[32]Z. Ni, M. Yang, Z. Ling, J. N. Wu and J. Luo, ”Real-Time Detection of Malicious Behavior in Android Apps,” In proc. Int. Conf. on Advanced Cloud and Big Data (CBD), Chengdu, 2016, pp. 221-227.

[33]K. M. Ting and I. H. Witten. Issues in Stacked Generalization. Journal of Artificial Intelligence Research, 10, May 1999, pp. 271-289.

[34]L. Breiman. Random forests. Machine Learning, 45, 2001, pp 5-32.

[35]M. K Alzaylaee, S. Y. Yerima, S. Sezer” Improving Dynamic Analysis of Android Apps Using Hybrid Input Test Generation” In proc. Int. Conf. On CyberSecurity and Protection of Digital Services (Cyber Security 2017), London, UK, June 19-20, 2017.

[36]Park, J.S.; Youn, T.Y.; Kim, H.B.; Rhee, K.H.; Shin, S.U. Smart contract-based review system for an IoT data marketplace. *Sensors* 2018, *18*, 3577. [Google Scholar] [CrossRef] [PubMed]

[37]Risteska Stojkoska, B.L.; Trivodaliev, K.V. A review of Internet of Things for smart home: Challenges and solutions. *J. Clean. Prod.* 2017, *140*, 1454–1464. [[Google Scholar](https://scholar.google.com/scholar_lookup?title=A%20review%20of%20Internet%20of%20Things%20for%20smart%20home:%20Challenges%20and%20solutions&author=Risteska+Stojkoska,+B.L.&author=Trivodaliev,+K.V.&publication_year=2017&journal=J.+Clean.+Prod.&volume=140&pages=1454%E2%80%931464&doi=10.1016/j.jclepro.2016.10.006)] [[CrossRef](https://dx.doi.org/10.1016/j.jclepro.2016.10.006)]

[38]Walls, J.; Choo, K.K.R. A review of free cloud-based anti-malware apps for android. In Proceedings of the 14th IEEE International Conference on Trust, Security and Privacy in Computing and Communications, Helsinki, Finland, 20–22 August 2015; Volume 1, pp. 1053–1058. [[Google Scholar](https://scholar.google.com/scholar_lookup?title=A%20review%20of%20free%20cloud-based%20anti-malware%20apps%20for%20android&conference=Proceedings+of+the+14th+IEEE+International+Conference+on+Trust,+Security+and+Privacy+in+Computing+and+Communications&author=Walls,+J.&author=Choo,+K.K.R.&publication_year=2015&pages=1053%E2%80%931058&doi=10.1109/Trustcom.2015.482)] [[CrossRef](https://dx.doi.org/10.1109/Trustcom.2015.482)]

[39]Chen, H.; Su, J.; Qiao, L.; Xin, Q.; Chen, H.; Su, J.; Qiao, L.; Xin, Q. Malware Collusion Attack against SVM: Issues and Countermeasures. *Appl. Sci.* 2018, *8*, 1718. [[Google Scholar](https://scholar.google.com/scholar_lookup?title=Malware%20Collusion%20Attack%20against%20SVM:%20Issues%20and%20Countermeasures&author=Chen,+H.&author=Su,+J.&author=Qiao,+L.&author=Xin,+Q.&author=Chen,+H.&author=Su,+J.&author=Qiao,+L.&author=Xin,+Q.&publication_year=2018&journal=Appl.+Sci.&volume=8&pages=1718&doi=10.3390/app8101718)] [[CrossRef](https://dx.doi.org/10.3390/app8101718)]

[40]Dogru, B.; Kiraz, O. Web-Based Android Malicious Software Detection and Classification System. *Appl. Sci.* 2018, *8*, 1622. [[Google Scholar](https://scholar.google.com/scholar_lookup?title=Web-Based%20Android%20Malicious%20Software%20Detection%20and%20Classification%20System&author=Dogru,+B.&author=Kiraz,+O.&publication_year=2018&journal=Appl.+Sci.&volume=8&pages=1622&doi=10.3390/app8091622)] [[CrossRef](https://dx.doi.org/10.3390/app8091622)]

[41]Xiangyu Liu, Zhe Zhou, Wenrui Diao, Zhou Li, and Kehuan Zhang. An empirical study on android for saving non-shared data on public storage. In ICT Systems Security and Privacy Protection, pages 542–556. Springer, 2015.

[42]V Babu Rajesh, Phaninder Reddy, P Himanshu, and Mahesh U Patil. Droidswan: Detecting malicious android applications based on static feature analysis. Computer Science & Information Technology, page 163.

[43]Yajin Zhou and Xuxian Jiang. Dissecting android malware: Characterization and evolution. In Security and Privacy (SP), 2012 IEEE Symposium on, pages 95–109. IEEE, 2012.