International Journal of Advanced Development in Science and Technology
Volume : 3 Issue : 03 April 2021 www.ijadst.com ISSN No: 2582-1059

CYBER THREAT ANALYSIS ON ANDROID APPS

[1] K. Lakshmi Priya, [2] P.Swetha, [3] S.Priskilla Manonmani,
[1,2] Final Year Students, [3] Assistant Professor,
[1,2,3] Department of Information technology,
[1,2,3] Meenakshi Sundararajan Engineering College,Chennai-21
[1] lakshmipriya3004@gmail.com

ABSTRACT

In recent years, the usages of smart phones are increasing steadily and also growth of Android application users are increasing. Due to growth of Android application user, some intruder is creating malicious android application as tool to steal the sensitive data and identity theft / fraud mobile bank, mobile wallets.

There are so many malicious applications detection tools and software are available. But an effectively and efficiently malicious applications detection tools needed to tackle and handle new complex malicious apps created by intruder or hackers. In this paper we came up with idea of using machine learning approaches for detecting the malicious android application. First we have to gather dataset of past malicious apps as training set and with the help of Support vector machine algorithm and decision tree algorithm make up comparison with training dataset and trained dataset we can predict the malware android apps up to 93.2 % unknown / New malware mobile application.

Keywords: ***dataset, machine learning algorithm- ensemble methods, python, predicting the accuracy of result, malware, threat analysis***

1. INTRODUCTION

Smartphones have become an integral part of our day-to-day life. New data for December 2018 shows that Android remains the most popular mobile operating system, with a worldwide market share of 75.16%. With over one million Android applications in major app stores, applications such as We Chat, TikTok, and mobile banking applications are used in our daily life and continue to play an increasingly important role.

Most of these applications have access to users' private information such as their location, credit card, and contact information. Almost all applications access the users' private data, although this provides users with better personalized services. It may also result in information leakage of private data and economic loss. Currently, static analysis and dynamic analysis are the two main types of detection methods. Each approach has its merits and shortcomings. The static analysis methods such as Kirin, PApriori and DREBIN analyze applications without executing the program requiring low overheads. However, the methods cannot defend against antide compiling and obfuscation.

With malware being rapidly evolving, the machine learning method is used to perform Android malware detection. Consequently, gathering features that better represent malicious behavior as the features of machine learning is beneficial to improve the performance of malware detection.

By itself, Android has several security mechanisms in its different layers. The permission mechanism applied in the application layer is an important defence mechanism to protect sensitive resources on the Android platform.

Applications must declare dangerous permissions to access sensitive data. Several studies have investigated Android malicious applications based on the declared permissions, using permission-based methods. Although these methods avoid high overhead, they consider the declared permissions as the features of machine learning, which cannot truly reflect the difference between benign applications and malicious applications.

Thus, they cannot detect malicious applications that declare only a few, or dangerous permissions, which are also always declared by benign applications. Compared with permissions as features, application programming interfaces (APIs) represent the entire picture of application behavior provided by the Android system.

DroidAPIMiner exploits data flow analysis to extract the numbers of APIs used in malicious applications and benign applications to analyze the difference between them, which is similar to these methods. In general, the API feature set contains a very large number of features, and therefore, detection methods need extra time to extract the API features and to train detection models. It is worth noting that there is a corresponding relation between permissions and APIs.

In this paper, we present FDP, a lightweight Android malware detection method to mine hidden patterns of malware. According to previous studies, there is considerable difference between malicious applications and benign applications in terms of the declared permissions.

In addition, some dangerous permissions declared for different components reflect the purpose of the developers. It can be used to distinguish different purposes of the same dangerous permission. These features can represent the difference between benign applications and malicious applications. In terms of efficiency, FDP uses static methods to gather all features and analyzes an application in a reasonable time.

2. PROPOSED SYSTEM

In proposed paper, we implements SIGPID, Significant Permission Identification (SIGPID).The goal of the sigid is to improve the apps permissions effectively and efficiently. This SIGPID system improves the accuracy and efficient detection of malware application. With the help of machine learning algorithms such as SVM and Decision Tree algorithms make a comparison between training dataset and trained dataset .Support vector machine algorithms act as a classifier which is used to classify malicious application and benign app.

We first use SVM and a small dataset to test our proposed MLDP model.

SVM determines a hyper plane/matrix that separates both classes with a maximal margin based on the training dataset that includes benign and malicious applications. In this case, one class is associated with malware, and the other class is associated with benign apps. Then, we assume the testing data as unknown apps, which are classified by mapping the data to the vector space to decide whether it is on the malicious or benign side of the hyper plane. Then, we can compare all analysis results with their original records to evaluate the malware detection correctness of the proposed model by using SVM. In order to show applicability and scalability of MLDP, we employ 67 commonly known machine learning algorithms and enlarge our dataset.

We compare the results between malware detection rate using all identified 135 permissions (baseline) and malware detection using MLDP for each supervised machine learning algorithm. We observe that, by analyzing all permissions, machine learning algorithms with a tree structure, usually build better malware detection compared to others. Among all the machine learning algorithms with a tree structure, our method works best on decision tree, which is a very common classification method in machine learning. Decision tree should use training dataset to build a classifier to decide which class the testing data should be, requires a lot of pre-processing work before the classifier was built. When there are too many attributes of training dataset, decision tree hits a poor accuracy and the training phase tend to take more time and memory. So, the pruning of the decision tree is imperative, which is consistent with our MLDP. Consequently, it is more advantageous to use MLDP to perform malware detection as it can be as effective while notably conserving time and memory.

Significant Permission Identification for Android Malware Detection. (SIGPID):

The goal of Significant Permission Identification (SIGPID) system is to achieve high malware detection accuracy and efficiency while analyzing the minimal number of permissions. To achieve this goal, our system extracts permission uses from application packages, but instead of focusing on all the requested permissions, SIGPID mainly focuses on permissions that can reliably improve the malware detection rate. This approach, in effect, eliminates the need to analyze permissions that have little or no significant influence on malware detection effectiveness. In a nutshell, SIGPID prunes permissions that have low impacts on detection effectiveness using multi-level data pruning to reduce analysis efforts.

Our system consists of three major components, designed based on real-time data analysis: (i) permission ranking with negative rate; (ii) support based permission ranking; and (iii) permission mining with association rules. After pruning, SIGPID employs supervised machine learning classification methods to identify potential Android malware. Finally, SIGPID reports malware detection summary to the analysts.

Multi-Level Data Pruning (MLDP)

The first component of SIGPID is the multi-level data pruning process to identify significant permissions to eliminate the need of considering all available permissions in Android. No app requests all the permissions, and the ones that an app requests are listed in the Android application package (APK) as part of manifest.xml. When we need to analyze a large number of apps (e.g., several hundred thousand), the total number of permissions requested by all apps can be overwhelmingly large, resulting in long analysis time. This high analysis overhead can negatively affect the malware detection efficiency as it reduces analyst productivity. We propose three levels of data pruning methods to filter out permissions that contribute little to the malware detection effectiveness. Thus, they can be safely removed without benign apps malicious.

3.ALGORITHM ENSEMBLE

Ensemble methods are meta-algorithms that combine several machine learning techniques into one predictive model in order to decrease variance (bagging), bias(boosting), or improve predictions(stacking). Ensemble methods can be divided into two groups:

- Sequential ensemble methods where the base learners are generated sequentially (e.g. AdaBoost). The basic motivation of sequential methods is to exploit the dependence between the base learners. The overall performance can be boosted by weighing previously mislabelled examples with higher weight.
- parallel ensemble methods where the base learners are generated in parallel (e.g. Random Forest). The basic motivation of parallel methods is to exploit independence between the base learners since the error can be reduced dramatically by averaging.

Most ensemble methods use a single base learning algorithm to produce homogeneous base learners, i.e. learners of the same type, leading to homogeneous ensembles. There are also some methods that use heterogeneous learners, i.e. learners of different types, leading to heterogeneous ensembles. In order for ensemble methods to be more accurate than any of its individual members, the base learners have to be as accurate as possible and as diverse as possible.

BAGGING

Bagging stands for bootstrap aggregation. One way to reduce the variance of an estimate is to average together multiple estimates. For example, we can train M different trees on different subsets of the data (chosen randomly with replacement) and compute the ensemble:

Bagging uses bootstrap sampling to obtain the data subsets for training the base learners. For aggregating the outputs of base learners, bagging uses voting for classification and averaging for regression.

BOOSTING

Boosting refers to a family of algorithms that are able to convert weak learners to strong learners. The main principle of boosting is to fit a sequence of weak learners− models that are only slightly better than random guessing, such as small decision trees− to weighted versions of the data. More weight is given to examples that were misclassified by earlier rounds. The predictions are then combined through a weighted majority vote (classification) or a weighted sum (regression) to produce the final prediction. The principal difference between boosting and the committee methods, such as bagging, is that base learners are trained in sequence on a weighted version of the data.

STACKING

Stacking is an ensemble learning technique that combines multiple classification or regression models via a meta-classifier or a meta-regress or. The base level models are trained based on a complete training set, then the meta-model is trained on the outputs of the base level model as features.

The base level often consists of different learning algorithms and therefore stacking ensembles are often heterogeneous.

SUPPORT VECTOR MACHINE

Support Vector Machine or SVM is one of the most popular Supervised Learning algorithms, which is used for Classification as well as Regression problems. However, primarily, it is used for Classification problems in Machine Learning. The goal of the SVM algorithm is to create the best line or decision boundary that can segregate n-dimensional space into classes so that we can easily put the new data point in the correct category in the future. This best decision boundary is called a hyper plane.

DECISION TREE ALGORITHM

Decision Tree algorithm belongs to the family of supervised learning algorithms. Unlike other supervised learning algorithms, the decision tree algorithm can be used for solving regression and classification problems too.

The goal of using a Decision Tree is to create a training model that can use to predict the class or value of the target variable by learning simple decision rules inferred from prior data(training data). In Decision Trees, for predicting a class label for a record we start from the root of the tree. We compare the values of the root attribute with the record's attribute. On the basis of comparison, we follow the branch corresponding to that value and jump to the next node.

NAIVE BAYES

Naive Bayes classifiers are a collection of classification algorithms based on Bayes' Theorem. It is not a single algorithm but a family of algorithms where all of them share a common principle, i.e. every pair of features being classified is independent of each other.
Gaussian Naive Bayes classifier

In Gaussian Naive Bayes, continuous values associated with each feature are assumed to be distributed according to a Gaussian distribution. A Gaussian distribution is also called Normal distribution.

4.MODULE

DATA COLLECTION

ML depends heavily on data. It's the most crucial aspect that makes algorithm training possible and explains why machine learning became so popular in recent years. But regardless of your actual terabytes of information and data science expertise, if you can't make sense of data records, a machine will be nearly useless or perhaps even harmful.
Data collection is the process of gathering and measuring information from countless different sources. In order to use the data we collect to develop practical artificial intelligence (AI) and machine learning solutions, it must be collected and stored in a way that makes sense for the business problem at hand.

In a nutshell, data preparation is a set of procedures that helps make your dataset more suitable for machine learning. In broader terms, the data prep also includes establishing the right data collection mechanism. And these procedures consume most of the time spent on machine learning.

To classify about the android apps we searched a dataset which consist of 1000 of applications . In 2020 we explored Android Genome Project

International Journal of Advanced Development in Science and Technology
Volume : 3 Issue : 03 April 2021 www.ijadst.com ISSN No: 2582-1059

(MalGenome), it is a dataset which was active from 2015 until the end of the year 2020, this set of malware has a size of 1260 applications, grouped into a total of 49 families. Today, we can find other jobs such as: Drebin, a research project offering a total of 5560 applications consisting of 179 malware families; AndrooZoo, which includes a collection of 5669661 applications Android from different sources (including Google Play); VirusShare, another repository that provides samples of malware for cyber security researchers; and Droid Collector, this is another set which provides around 8000 benign applications and 5560 malware samples, moreover, it facilitates us samples of network traffic as pcap files. So we collected the data set from kaggle .

PRE PROCESSING

Organize your selected data by formatting, cleaning and sampling from it.

Three common data pre-processing steps are:

1. Formatting
2. Cleaning
3. Sampling

Formatting: The data you have selected may not be in a format that is suitable for you to work with. The data may be in a relational database and you would like it in a flat file, or the data may be in a proprietary file format and you would like it in a relational database or a text file.

Cleaning: Cleaning data is the removal or fixing of missing data. There may be data instances that are incomplete and do not carry the data you believe you need to address the problem. These instances may need to be removed. Additionally, there may be sensitive information in some of the attributes and these attributes may need to be anonym zed or removed from the data entirely.

Sampling: There may be far more selected data available than you need to work with. More data can result in much longer running times for algorithms and larger computational and memory requirements. You can take a smaller representative sample of the selected data that may be much faster for exploring and prototyping solutions before considering the whole dataset.

FEATURE EXTRACTION

Next thing is to do Feature extraction is an attribute reduction process. Unlike feature selection, which ranks the existing attributes according to their predictive significance, feature extraction actually transforms the attributes. The transformed attributes, or features, are linear combinations of the original attributes. Finally, our models are trained using Classifier algorithm. We use classify module on Natural Language Toolkit library on Python. We use the labelled dataset gathered. The rest of our labelled data will be used to evaluate the models. Some machine learning algorithms were used to classify pre-processed data. The chosen classifiers were Decision Tree, Support Vector Machine. These algorithms are very popular in text classification tasks.

CLASSIFICATION

The purpose of this phase is to select different features for different classes by applying the information gain or gain ratio in order to identify relevant features for each binary classifier. The attribute with the highest normalized information gain is chosen to make the decision. The C4.5 algorithm then recourse on the smaller sub lists.

This algorithm has a few base cases.

- All the samples in the list belong to the same class. When this happens, it simply creates a leaf node for the decision tree saying to choose that class.
- None of the features provide any information gain. In this case, C4.5 creates a decision node higher up the tree using the expected value of the class.
- Instance of previously-unseen class encountered. Again, C4.5 creates a decision node higher up the tree using the expected value.

EFFICIENCY CALCULATION

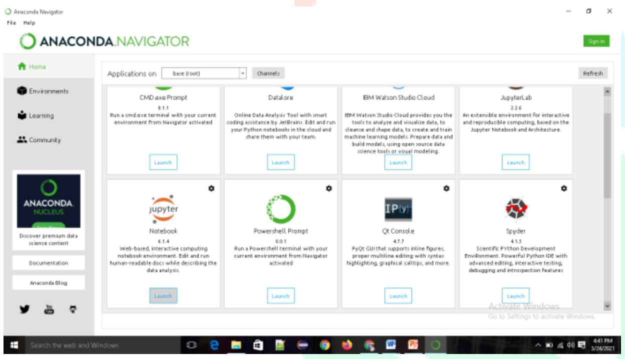
The effect of combining different classifiers can be explained with the theory of bias-variance decomposition. Bias refers to an error due to a learning algorithm while variance refers to an error due to the learned model. This is why the idea emerged of combining both in order to profit from the advantages of both algorithms and obtain an overall error reduction.

The concept of bagging (voting for classification, averaging for regression-type problems with continuous dependent variables of interest) applies to the area of predictive data mining, to combine the predicted classifications (prediction) from multiple models, or from the same type of model for different learning data. It is also used to address the inherent instability of results when applying complex models to relatively small data sets. Suppose your data mining task is to build a model for predictive classification, and the dataset from which to train the model (learning data set, which contains observed classifications) is relatively small. It could repeatedly sub-sample (with replacement) from the dataset, and apply, for example, a tree classifier to the successive samples.

- In practice, very different trees will often be grown for the different samples, illustrating the instability of models often evident with small datasets. One method of deriving a single prediction (for new observations) is to use all trees found in the different samples, and to apply some simple voting: The final classification is the one most often predicted by the different trees. Note that some weighted combination of predictions (weighted vote, weighted average) is also possible, and commonly used.

5. RESULT ANALYSIS

Opening Jupyter Notebook using Anaconda Navigator Analyzing the malware and benign apps in the trained data set



CMD.exe Prompt 0.1.1 Run a cmd.exe terminal with your current environment from Navigator activated

DataLore Online Data Analysis Tool with smart coding assistance by JetBrains. Edit and run your Python notebooks in the cloud and share them with your team.

IBM Watson Studio Cloud IBM Watson Studio Cloud provides you the tools to analyze and visualize data, to cleanse and shape data, to create and train machine learning models. Prepare data and build models, using open source data science tools or visual modeling.

JupyterLab 2.2.6 An extensible environment for interactive and reproducible computing, based on the Jupyter Notebook and Architecture.

Notebook 6.1.4 Web-based, interactive computing notebook environment. Edit and run human-readable docs while describing the data analysis.

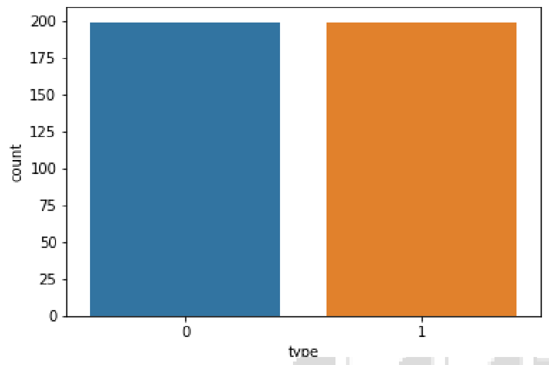
Powershell Prompt 0.0.1 Run a Powershell terminal with your current environment from Navigator activated

Qt Console 4.7.7 PyQt GUI that supports inline figures, proper multiline editing with syntax highlighting, graphical calltips, and more.

Spyder 4.1.5 Scientific PYthon Development EnviRonment. Powerful Python IDE with advanced editing, interactive testing, debugging and introspection features

Discover premium data science content

Activate Windows Go to Settings to activate Windows.



0-Benign 1-Malicious

Analyze the malware and benign apps in the test data set Tracking performance of the phone using Phone state

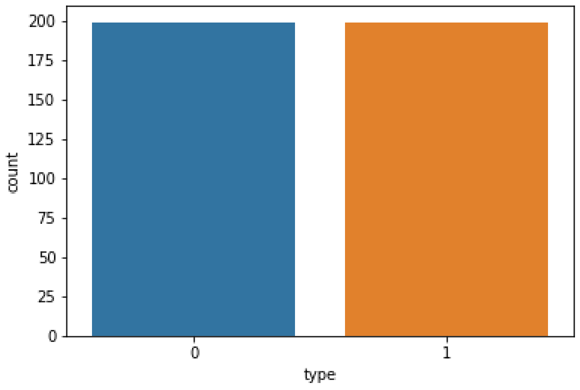
SUPPORT VECTOR MACHINE OUTPUT

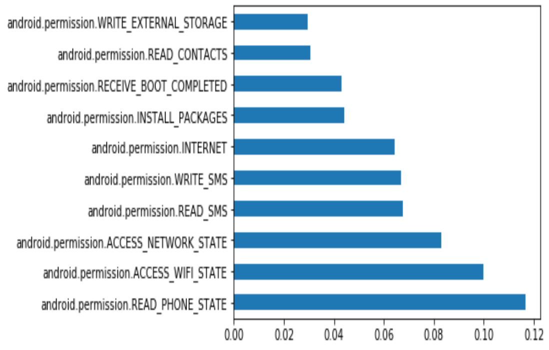
array([1, 0, 1, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1,

0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1,

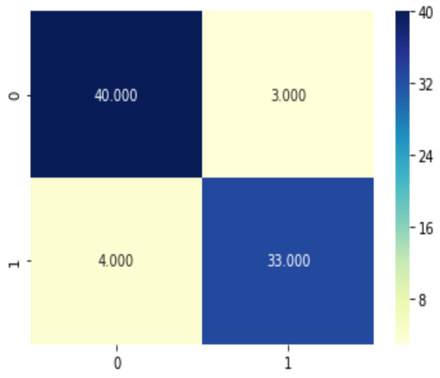
0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0,

0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0], dtype=int64)





Svm Confusion matrix

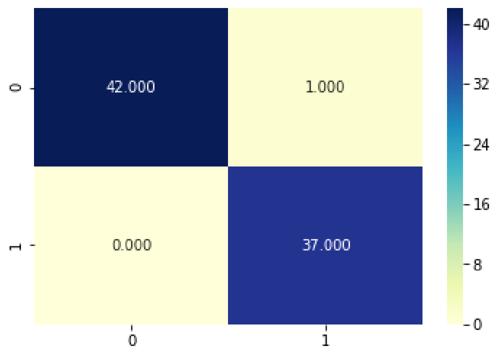


DECISION TREE OUTPUT

array([1, 0, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1,

```
        0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1,
0, 1, 1, 1, 1, 0, 1,
        0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 1,
0, 0, 0, 0, 0,
        0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 0],
dtype=int64)
```

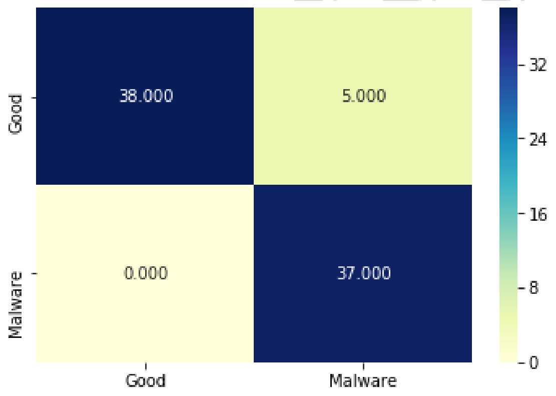
Decision Tree Confusion Matrix



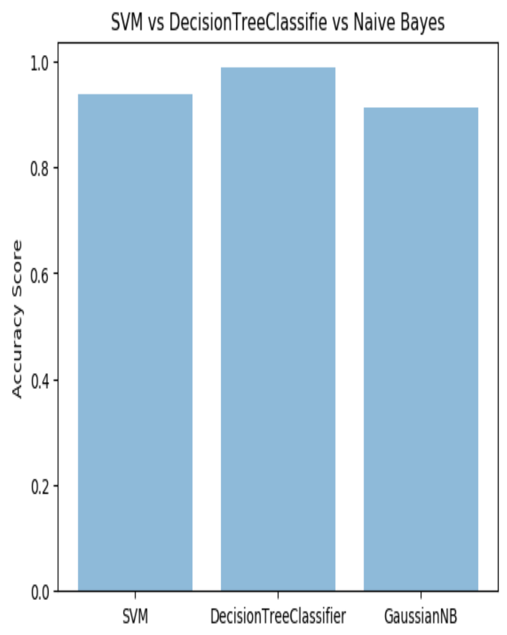
GAUSSIAN NAÏVE BAYES OUTPUT

```
array([1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0,
1, 1, 1, 1, 0, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1,
1, 1, 1, 1, 1,
        0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 0, 0, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1],
dtype=int64)
```

Gaussian Naïve Bayes Confusion Matrix



COMPARISON GRAPH



6. CONCLUSION

In static analysis of Android malware, machine learning algorithms have been used to train classifiers with features of malicious apps to build models that capable of detecting malicious patterns. Differently, our classification approach defines legitimate static features for benign apps as opposite to identifying malicious patterns. We utilize the features of the top rated apps in a specific category to define a profile of the common sets of features for that category.

7. FUTURE ENHANCEMENT

Our future work will consider three aspects. First, including other static features such as: functions calls in building the classification models to get a better understanding of the processes that apps may lunch in a way to increase the detection accuracy of the classifiers. Second, implementing the proposed solution on a large-scale level by building profile models for other categories and sub categories.By making this as a web based application user can easily keep track of the application details. Third, testing the feasibility of integrating our solution with dynamic detection techniques by profiling dynamic features for each category; dynamic features like system calls, network connections, resources' usage, and etc and allowing the each and every user to access the portal easily.

8.REFERENCES

[1] Androguard usage. https://code.google.com/p/androguard/wiki/Usage. Accessed April 24, 2015. [2] Android - statistics & facts — statista. http://www.statista.com/topics/876/ android/. Accessed April 19, 2015.

[3] Android and ios continue to dominate the worldwide smartphone market with android shipments just shy of 800 million in 2013, according to idc. http://www.idc. com/getdoc.jsp?containerId=prUS24676414. Accessed April 19, 2015.

[4] Application fundamentals — android developers. http://developer.android.com/ guide/components/fundamentals.html. Accessed April 19, 2015.

[5] Are — download/installation. https://redmine.honeynet.org/projects/are/ wiki. Accessed April 28, 2015.

[6] SpyDroid: A Framework for Employing Multiple Real-Time Malware Detectors on Android-Shahrear Iqbal, et al 2018

[7] Samadroid: a novel 3-level hybrid malware detection model for android operating system saba arshad1, munam a. shah1, abdul wahid1, amjad mehmood2,
houbing song 3, (senior member, ieee), and hongnian yu4, 5

[8] Meta-Feature Classification to Explore Automatic Detection of Malware Using Segmentation Method Chandra Sekhar Vasamsetty, Siva Sankar Chandu, Janakidevi Maddala

[9] DroidMat: Android Malware Detection through Manifest and API Calls Tracing 2012 Seventh Asia Joint Conference on Information Security

[10] Improving Dynamic Analysis of Android Apps Using Hybrid Test Inputn Generation Alzaylaee, M. K., Yerima, S. Y., & Sezer, S. (2017). Improving Dynamic Analysis of Android Apps Using Hybrid Test Input Generation