Maldroid: Dynamic Malware Detection using Random Forest Algorithm

Gowtham Sethupathi, Swapnil Siddharth, Vikash Kumar, Pratyush Kumar, Ashwani Yadav

Abstract*: Android platform has become one of the most widely used operating system for smart-phones with the number of users increasing day by day. The number of applications made for Android is also rising. Consequently, with the advantages of the apps, malware tend to come into the scene with an opportunity to harm the operating system as well as steal sensitive data available on the phone. To address a solution to this problem, a number of static and dynamic analysis tools and techniques have been introduced to distinguish between millions of Android apps in the marketplace and malicious apps. In this paper we study the benign Android apps, mine the pattern to classify benign apps from malicious apps. We also create an automated malware detection system, Maldroid, a program to find if an application is malicious or not. The program was tested with a large dataset of around benign apps and malicious app. Experimental results show that Maldroid is capable of detecting malware with relatively high F1 score of 98 percent.

Index Terms*: Android applications, malware detection, permission- related APIs, random forests, software security.

I. INTRODUCTION

Android has become one of the most widely used operating system today for smart phones and tablets. According to reports, the number of users is around 5 billion as of 2018. The reason for this large number of users is due to wide variety of applications such as camera, web browser, games, etc. available on it. However, these smart phones also contain many sensitive data of users which can be stolen through malwares. Since, most of the users rely on smart phones as their primary assistant, their personal data and information becomes vulnerable. According to a Symantec report, the number of malwares targeting Android operating system has increased significantly over the recent years. Android users often cannot perceive if the app being installed in their phones are a trustworthy app or malicious app. Whenever a user installs an app, they are under the risk of installing a malware. In desktop applications, the apps don't have the privileges to access in the sensitive data of computer. However, mobile apps can have the privileges after declaring it while installing. They can access sensitive information such as contact list, SMS, GPS, etc. Most of the apps tend to access the wide range of information and resources available on the phone in order to fulfil its job. Users who are in a hurry to use the app, tend to grant the permission to these apps.

Unexpectedly, it tends to provide opportunity for malware to hijack and steal private and sensitive information.

According to McAfee 2016 report, there are 13 million mobile malware samples and it recorded a 72 percent in-crease in new mobile malware samples as to the last quarter. Data leak is one the major task of these malwares along with stealing users contacts, location history, in some cases money. Since, most of the population is switching to digital banking via smart phones instead of cash, it has become a challenge to protect the bank details of customer from malicious activity on the line. There have been numerous cases of stealing credit card details by many underground groups.

To address these problems and to provide a reliable solution, we perform a thorough study on Android malwares as well as benign apps. Static and dynamic Analysis have been thoroughly studied to find out the best method for detection. Furthermore, using machine learning algorithms, a system is prepared which can effectively detected any malware with high accuracy.

II. LITERATURE REVIEW

A. Andlantis: Large-scale Android Dynamic Analysis

This paper was prepared by Michael Bierma, Eric Gustafson, Jeremy Erickson, David Fritz, Yung Ryn Choe which focuses on analysis of Android applications for ma- licious behavior by building Andlantis. It is a scalable dynamic analysis system which processes more than 3000 apps per hour. After which it provides relevant data to understand anomalous application behavior. The dataset consists of 1261 malware samples along with benign samples. Andlantis is also shown to be robust to node failures, and has a very low job failure rate of less than 3 percent.

B. DREBIN: Effective and Explainable Detection of Android Malware in Your Pocket

In this paper, prepared by Daniel Arp, Michael Spreitzenbarth, Malte Hubner, Hugo Gascon, Konrad Rieck, a lightweight method for detection of Android malwares has been proposed. It performs broad static analysis and gathers features of an application. The collected features are embedded in a joint vector space, which is used to indicate pattern for finding malware. This is run on smart phones, and the method takes 10 seconds for analysis.

Revised Manuscript Received on April 06, 2019.

Gowtham Sethupathi, Computer Science & Engineering, SRM Institute of Science and Technology, Chennai, Tamil Nadu, India.

Swapnil Siddharth, Computer Science & Engineering, SRM Institute of Science and Technology, Chennai, Tamil Nadu, India.

Vikash Kumar, Computer Science & Engineering, SRM Institute of Science and Technology, Chennai, Tamil Nadu, India.

Pratyush Kumar, Computer Science & Engineering, SRM Institute of Science and Technology, Chennai, Tamil Nadu, India

Ashwani Yadav, Computer Science & Engineering, SRM Institute of Science and Technology, Chennai, Tamil Nadu, India

C. Malware Detection in Android App Using Static and Dynamic

This paper was prepared by Priyanka Tate, Rachana Sonawane and Sagar Shinde. The objective of this paper was to build a deep learning-based Android malware detection engine named as DroidDetector which identifies whether an application is malware or not. The system performs an in-depth analysis for features that deep learning exploits to distinguish malware from benign app.

D. A comparative study of static, dynamic and hybrid analysis techniques for android malware detection

This paper was prepared by Prof. Vidhya Rao, Khushboo Hande. The paper focuses on different types of malware detection techniques and gives an analysis of every method. As a single approach is not enough for detecting malware, a multiple approach has been suggested such as static, dynamic and hybrid analysis (which is a combination of static and dynamic). The functionality of each is studied along with comparative study between them. Finally, hybrid analysis, the most efficient one is highlighted and suggested.

III. METHODOLOGY

In this section we discuss some of the state-of-the-art detection techniques used for detecting malwares of mobile phones. They broadly fall into 2 categories which are static technique and dynamic techniques.

A. Static Techniques

Static methods focus on source code of the application to find and classify the app accordingly without the need of the app to be executed. DroidRange, Flowdroid and Debrin are some system which used static analysis. Static techniques are further classified into more classes:

- Signature-based approach: This approach extracts the semantic pattern and creates a unique signature. An app is classified as malware if its signature matched with existing signature. This approach is quite fast but can be bypassed by code obfuscation. It also fails against unseen variants of malware and requires a frequent update of malware signatures.
- Permission-based analysis: Every app when it is being installed, requires permission from users to access rights. By default, an app has not permission to access user's data and system access. Users have to allow the app to access the requested permissions. These techniques focus on classifying using the permission asked by the apps which makes it fast but it does not analyses any other file.

B. Dynamic Techniques

In the dynamic analysis technique, the application is analyzed while being executed in the environment. TaintDroid, VetDroid and DroidScope are some of the detection system which falls into this category. The dynamic analysis are further classified according to the behavior of the detection mechanism.

- Anomaly Based: This method keeps a track of different system parameters and components of device. To detect the presence of a malware, the behavior of the device is monitored. This includes, battery level, CPU usage, etc.

These behaviors are taken and applied to an algorithm which performs the classifications.

- Taint Analysis: This tracks multiple sources of sensitive data and finds out any leakage in application. The tool figures the flow of sensitive data in the device. It is efficient in tracking the flow of data but cannot perform control flow tracking.

C. Machine Learning Approach:

For classification of malware, machine learning techniques are widely used. The permissions which are formed to protect data of user and system directly demonstrates the sensitive behavior of apps. By analyzing the permission usage of hundreds of benign and malicious apps, the abnormal behaviors can be studied which are be further used to classify malware and benign apps. The permission-related APIs can be used for the system. Further, random forests, a popular machine learning approach us used to classify malware and benign apps. Random forest basically is used for classification and regression. It consists of set of binary decision trees. Through the training of multiple decision trees, the algorithm combines results from these trees with voting approach.

IV. DATASET COLLECTION

A major part of solving any problem with machine learning is gaining proper dataset for the training model. Getting the proper data consists of gathering or identifying the data that correlates with the outcomes the system wants to predict. In order to find the pattern of malwares, we have to study the behavior of malicious and benign Android apps in the real world. In this section, we have described two sets of Android apps as wells as their characteristics.

A. Benign apps

In order to collect the benign apps, we have used Google Play, which the official Android app marketplace. The first dataset consists of 28 thousand apps randomly collected from the period of May to December 2015. There are different kind of apps with different usage such as games, tools, etc. We have also downloaded the top 100 most popular free apps. The second dataset consists of around 2 thousand apps.

B. Malicious apps

The malicious apps have been collected from 2 sources:

• The first set consists of around 24 thousand apps which was obtained from VirusShare. These malwares were generated from period of May 2013 to March 2014. Many of these apps could be decompiled properly so there were around 14 thousand apps left for data set.

• The second dataset consists of malwares collected from Contagio. These contain around 500 malicious apps collected in the period from October 2010 to January 2016.

Totally, the malware dataset contains around 15 thousand

apps.

V. SYSTEM ARCHITECTURE

In this section, we describe the proposed system architecture of Maldroid with the help of Figure 1. The architecture consists of four major parts:

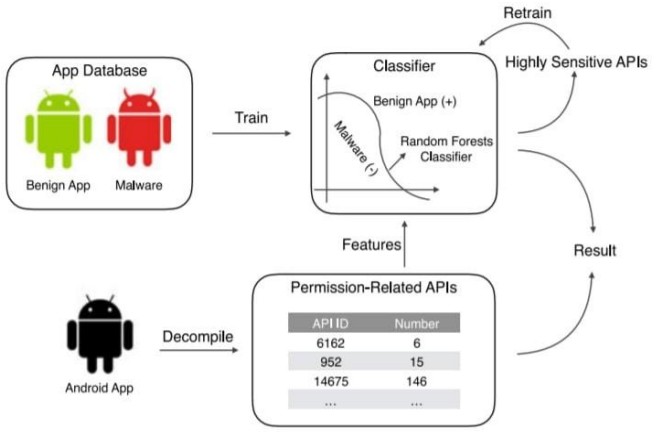

Figure 1: System Architecture

A. Collection of Dataset

The first part is the collection of datasets. A large collection of malware apps as well as benign app have been collected to form the dataset for training the model using machine learning algorithm. Both the malicious app and benign app dataset are merged to form a large dataset. The benign app dataset consists of around 30 thousand apps whereas malicious apps dataset has around 15 thousand apps.

B. Training of Dataset

The generated dataset is used for training the model with given parameters using random forest algorithm. Data Prepossessing is an important part of this process where the data must be organized. Also, the relation between these datasets must be found to extract features. Data cleaning is also a major part, to find out data with missing features and values. A lot of malicious apps collected from VirusShare could not be decompiled properly, thus they were rejected and fresh apps were collected from Contagio.

C. Extracting of Feature

The feature extraction is important for prediction and selected meticulously in order to perform faster computation and low memory consumption. With the collected malicious and benign apps, the features are taken from source codes of decompiled files. The installation package of Android apps is basically the .apk file, that can be decompiled using Apktool. It tends to recover main files organizing source codes in a particular way in folder. Then the methods implemented in the source fall in two formats. The first part android/net/ConnectivityManager gives the package of the invoked method whereas the second part is the target method, getActiveNetworkInfo(), which is used in the app. Using this we traverse through the decompiled source codes to extract employed APIs of the target app that forms the initial feature set. For both malicious and benign app datasets, the numbers of permission-related APIs are extracted as the features to train the classifier.

D. Deployment of Random Forest Algorithm

Now after obtaining the extracted features, the random forest algorithm is used for classification. During the training process, a set of labels is set to determine the type of each app.

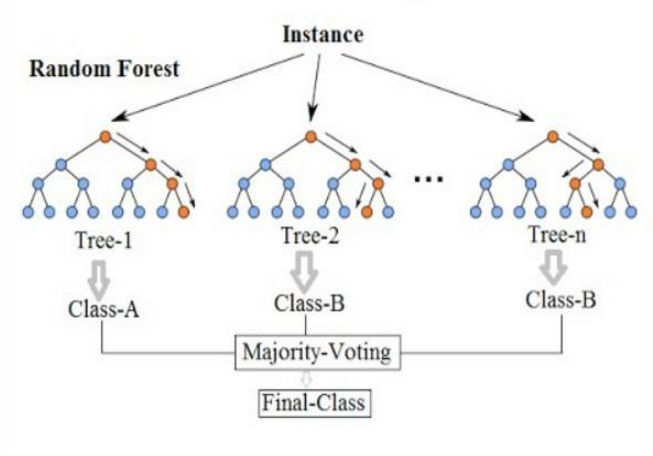

Figure 2: Random Forest Algorithm

Thus, 1 denotes malware and 0 denotes benign apps. The construction of random forests consists of a collection of decision trees where the number of decision trees is set manually. Here every decision tree is developed in a top-down approach initially starting from the root. At each node of the decision tree, it splits the training set into two subsets with different labels by minimizing the uncertainty of the class labels. After the training process is completed, the parameters of random forests at each node of each decision tree are set and have the capability of classifying apps.

VI. MODULE DESCRIPTION

In this section we discuss the various modules which run the detection system. The modules flowchart diagram is Figure 3. The following are the modules of the proposed system:

1. **Training of Dataset:** From the collected dataset the extraction of API from Android takes place and Java source code is decompiled to get function calls. The tool here used is Apktool. The generated dataset are used to train the random forest model with our given parameters and decided layers. The data is pre-processed and the files which could not be de-compiled are removed. Both the benign app set and malicious app set are used for training the model.
2. **Feature Extraction:** From the combination of permissions as ML input trained dataset of benign apk files as wells as malicious apk files, the main task is to find the malicious patterns of permission. The permission-based features are extracted, these features have been used for differentiating from malicious permission requests and benign request. The feature extraction is important for prediction. the better the features the faster the computation as well as low memory consumption. after the de-compilation brings out two formats in a folder. The first part is

android/net/ConnectivityManager gives the package of the invoked method whereas the second part is the target method, getActiveNetworkInfo(), which is used in the app. This is used to traverse through the decompiled source codes to extract employed APIs of target app that forms the initial features sets. The model is prepared from both the malicious and benign app datasets, the number of permission-related APIs are extracted as feature to form the model which can be used to predict any app from the test dataset.

3. **Random Forest Algorithm**: After extracting the features, random forest algorithm is used for classification. The name if we break down the word, it consists of 'forest' which consist of group of decision trees, and the word 'random' comes because we are doing random sampling. On applying this algorithm on a data set, it takes a subset of the data as training set and clusters the data into groups and subgroups. On connecting the data points to groups and sub-groups we get a structure resembling a tree, called decision tree. The algorithm then prepares a number of trees, resembling a forest. But each tree is different, as for each split in the tree, the variables are chosen randomly. The remaining data set, apart from the training set is used for predicting the tree in forest which makes best classification of data points and the tree having most predictive power is shown as output. Then, a set of labels is set to determine the type of each app where 1denotes malware and 0 denotes benign apps. At each node of the decision tree, it splits the training set into two subsets with different labels by minimizing the uncertainty of the class labels.

4. **Prediction**: After training of dataset and finding permission-based features, a model is prepared. A relatively unknown application is sent as new data for predicting malicious of benign, the parameters of random forests at each node of each decision tree are set and have the capability of classifying apps. When any new app is brought in as input, it is decompiled using the Apktool. As discussed above the source codes of the app are all stored in smali folder. Then all the permission-related APIs, including the number of call sites for each API, as the initial features. The highly sensitive APIs are able to be mined from thousands of permissions- related APIs by training the random forests classifier. Thus, here these highly sensitive APIs are used as the final features of the any new app. These features are then employed as the input to obtain the result of the app type either as malware or benign

VII. RESULT

In this section we discuss the evaluation metrics adopted as well as the results compared with other detection techniques. Many experiments are conducted for Maldroid. The comparison with other state-of-the-art approaches are also discussed along with illustrations.

A. Evaluation Metrics

To evaluate the performance of malware detection system,

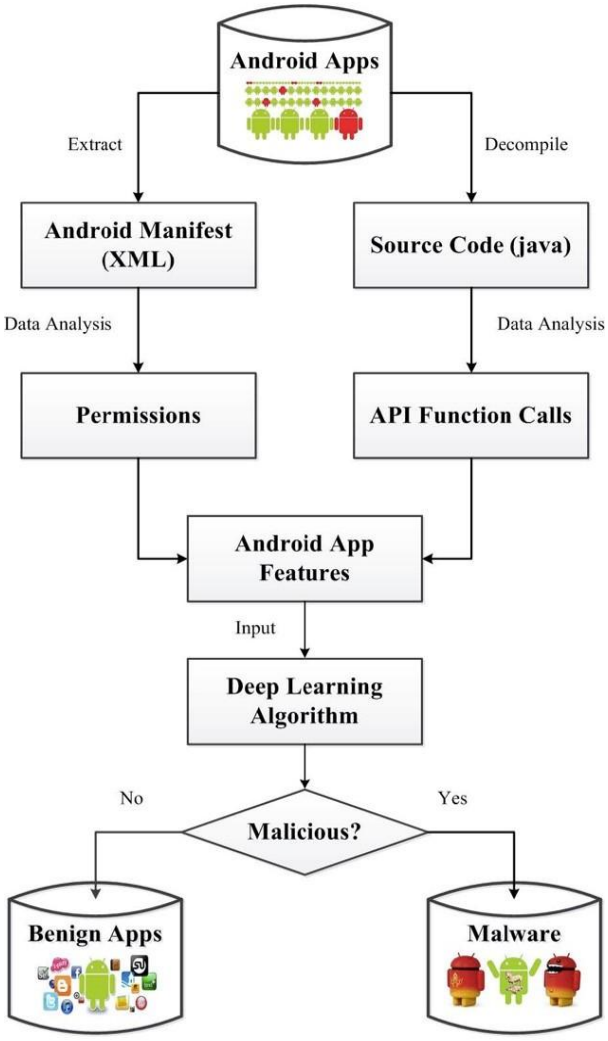

Figure 3: Module Description

the precision and recall metrics has been used here. In our evaluation test we have kept malicious apps as positive samples and benign apps as negative samples, thus, we first provide three types of values:

1) (tp: true positive): The number of malicious apps that are correctly identified as malicious apps.
2) (fp: false positive): The number of benign apps that are incorrectly identified as malicious apps.
3) (fn: false negative): The number of malicious apps that at are incorrectly identified as benign apps. Therefore, the metrics precision and recall can be calculated as follows:

precision = tp/tp + f p
recall = tp/tp + f n(5)
F 1 = 2 precision recall/precision + recall

The value of precision falls in the interval of [0, 1], where the large value indicates the correctness of the detection system. The recall equation denotes about how many malicious apps which have been identified by detection system are true malware. The recall value also lies between [0,1]. The F1 score equation is the average of

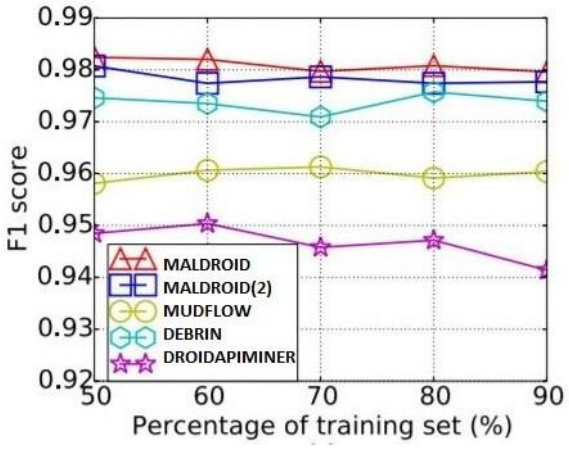

Figure 4: Score Comparision

precision and recall. The value of F1 score is also in the interval of [0, 1]. In order to successfully detect malware, our objective is on the correctness of the identification of malicious apps instead of benign apps. Therefore, precision, recall, and F1 score of malwares are adopted as evaluation metrics in the experiments.

B. Results

We test out Maldroid with other state-of-the-art approaches such as MUDFLOW, DroidAPIMiner and DE- BRIN. We use a dataset which consists of around 2 thousand benign apps and 13 thousand malware samples. There is no interaction between the dataset adopted for different detection systems. For all experiments, random set datasets are taken, and the some are left which will be used as the test set. A number of times the experiment is run in order to get the average results. The results have been compared in the Figure 3.

On comparison with existing state-of-the-art approaches, Maldroid demonstrates its effectiveness in identifying mal-ware. The highest recall rate achieved by Maldroid is 0.9963, that means that only 51 out of 13 840 malicious apps are not detected by the system

VIII. CONCLUSION

In order to fight effectively against malwares in Android, we study malicious apps and benign apps in real world. We mine the hidden patterns of Malware. Our previous research work focused on permissions, intents, sensitive resources, etc. and a very few of them addressed the malware detection from API point-of-view. In order to address this gap, we analyze the behavior of malicious apps on API usage while comparing with benign apps. We train the extracted highly sensitive APIs using random forest classifiers. A dynamic malware detecting system has been prepared, Maldroid. This system performs experiments on large datasets of malicious and benign apps. MalDroid achieves the F1 score of 98.24 percent. It is much effective when compared with other state-of-the-art approaches such as DEBRIN and MUFLOW.

While Maldroid seems efficient, there are lot of places where it can be improved to make pattern mining effective as well as malware detection. We can include a larger dataset in the future works, which will avoid the overfitting problem. We also have to include more benign apps to study different kind of apps which are normally used which will help in the detection of benign apps. In future work, we will focus on improving the capability of Maldroid and help the Android app marketplaces to fight against malware efficiently.

ACKNOWLEDGMENT

Our project on dynamic malware detection using machine learning algorithms, we would like to thank our project guide Prof. Gowtham Sethupathi for helping us throughout our project and giving as important time to time to execute our project.

REFERENCES

1. "McAfee Labs Threats Report March 2016," [Online]. Available: http://www.mcafee.com/us/resources/reports/rp-quarterly-threatsmar-2016.pdf
2. S. Rasthofer, I. Asrar, S. Huber, and E. Bodden, "How current android malware seeks to evade automated code analysis," in Proc. 9th Int. Conf. Inf. Security Theory Practice, 2015, pp. 187–202
3. V. Avdiienko et al., "Mining apps for abnormal usage of sensitive data," in Proc. 37th IEEE Int. Conf. Softw. Eng, 2015, pp. 426–436.
4. D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, and K. Rieck,
5. "DREBIN: Effective and explainable detection of android malware in your pocket," in Proc. 21st Annu. Netw. Distrib. Syst. Security Symp.
6. H. Peng et al., "Using probabilistic generative models for ranking risks of android apps," in Proc. 19th ACM Conf. Comput. Commun. Security, 2012, pp. 241–252.
7. J. Wang, Ò Fundamentals of erbium-doped fiber amplifiers arrays (Periodical style Submitted for publication),Ó IEEE J. Quantum Electron., submitted for publication.
8. Y. Aafer, W. Du, and H. Yin, "DroidAPIMiner: Mining API-level features for robust malware detection in android," in Proc. 9th Int. Conf. Security Privacy Commun. Netw., 2013, pp. 86–103.
9. T. K. Ho, "Random decision forests," in Proc. 3rd Int. Conf. Document Anal. Recognit., 1995, pp. 278–282
10. L. Breiman, J. Friedman, C. J. Stone, and R. A. Olshen, Classification and Regression Trees. Belmont, CA, USA: Wadsworth, 1984.
11. J. Ross Quinlan, C4.5: Programs for Machine Learning. San Mateo, CA, USA: Morgan Kaufmann, 1993.
12. V. Avdiienko et al., "Mudflow," [Online]. Available: https://www.st.cs. uni-saarland.de/appmining/mudflow/

AUTHORS PROFILE

Gowtham Sethupathi is professor in SRM Institute of Science and Technology for CSE department. He has great knowledge in field of AI due to his teaching experience. Currently he is also involved in other research work with her students

Swapnil Siddharth is currently pursuing her B. Tech in CSE from SRM Institute of Science and Technology.

Vikash Kumar is currently pursuing her B. Tech in CSE from SRM Institute of Science and Technology.

Pratyush Kumar is currently pursuing her B. Tech in CSE from SRM Institute of Science and Technology.

Ashwani Yadav is currently pursuing her B. Tech in CSE from SRM Institute of Science and Technology.