Automated Malicious Android App Detection using Machine Learning Methods

[1] Tendai Munyaradzi Marengereke, [2] K. Sornalakshmi
[1]M.Tech Student, [2] Assistant Professor
[1]Information Security & Cyber Forensics, [2] Department of Information Technology
SRM University, Kattankulathur Campus, Chennai, India

***Abstract* - There has exponential been growth of mobile computing, and as the capabilities of smartphones has increased so have malware threats. Enterprises and individual users have extensively adopted the use of Android mobile devices, users can download apps from unofficial marketplaces which pose a security risk. We overview Android malware detection methods and showcase an architecture for a malware detection system as an in-cloud service. The architecture uses automated static methods to decompile apk source code and then utilizes machine learning methods to classify risky, malicious and benign apps according to permissions requested, and installation origin. We present our system and experimental results on a dataset of 300malware and 500 benign application. Our trained model provides a detection rate after evaluation of 89%.**

***Index Terms* - Android, Malware Detection, Classification, Machine Learning, Data Mining, Cloud Offloading.**

I. INTRODUCTION

Android has experienced rapid growth, and as a result so has malware on the platform. Android malware is divided into different categories, private data stealers, premium service apps and service disruption apps. New iterations of malicious apps keep appearing in the wild, such as the *Simplelocker* ransomware [1], which encrypts user information on the mobile and only releases the decryption key after the user has paid a certain fee. *Simplelocker [12]* connects to a command control server via *Tor*. [2] It is hard to keep up with new malware as they are made.

Many applications for detecting malware are available on the Android Play store, e.g. Avast, AVG, and there is also the official app verification service *GoogleBouncer* which checks for malicious apps uploaded to the official app Play Store. In the case that an app installed from the Play Store is malicious and has somehow bypassed the vetting process, it can be remotely uninstalled from the device, but by then damage could have already been done. Android OS itself contains vulnerabilities, such as the Fake ID vulnerability which can be abused by malware applications, allowing other apps to impersonate trusted and signed apps without any user notification [3].

Jian [4], found that the official bouncer had a 15.32% detection rate, and furthermore most application side loaded onto phones from unofficial channels were never detected. The open source attribute of Android, has become a bane to security attempts to circumvent malware distribution. Currently there are multiple app store which include official Google Play store [5], Amazon App Store [13], MoboGinie, Getjar amongst others, and his fragmentation increases attacks vectors.

In this work, we overview methods of anti-malware techniques on Android mobiles and present an integrated system for analyzing, detecting and reporting Android malware. Our system uploads applications and their data to a server, which automatically performs static analysis on the uploaded files and extracts a set of features to be used by machine learning model to classify malicious apks from benign files. We conducted training and testing with a limited dataset of 500 benign applications downloaded from the Google Play Store, as well as 300 malware gathered from various sources which include ContagioDump[6], Virusshare[7].

The remainder of this paper is structured as follows. Section 2 introduces the background literature relevant to this paper, this includes overview of Android security, malware detection techniques and the use of machine learning in malware detection. Section 3 overviews the methodology and design of our detection system. In Section 4 we evaluate our experimental results. Finally Section 5 discusses limitations and provides an outlook on future work and concludes the paper.

II. BACKGROUND

Google Android [8] is an open source Linux Based operating System, it currently dominates the current smartphone OS market. It currently accounts for approximately 84% of smartphone sales as of 2014 4th quarter see Fig 1. As it is open source it allows manufacturers to make changes to UI or other elements of the OS.

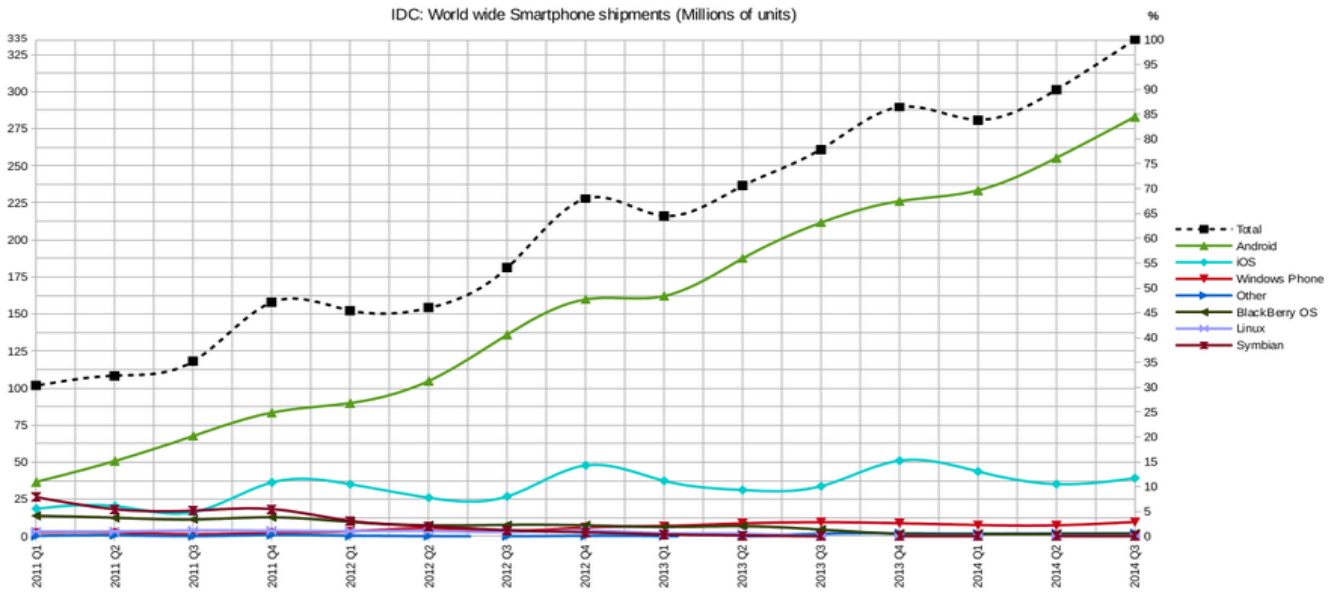

Figure 1 Worldwide Smartphone Shipments

Android OS executables are known as apps and are written in Java, the java class files are converted into dex btyecode and then packaged into Android PacKage (APK) file format: an archive file built on the ZIP file format. Apps are provided via different market places, where users can download and install applications onto their mobiles. Google Play Store, formerly known as Android Market is the official and largest Android app market with over 1,3millions unique applications. Android OS also allows users to install from 3rd party markets such as aforementioned Amazon App Store, amongst others. And this is where the danger of malware comes in. Android malware mainly comes in the form of malicious apps, and Android has been mainly targeted due to its popularity.

A. Android Security

Android security is based on Linux, the security mechanisms are offered in terms of; 1. Permissions, 2. Application Signing, 3. File rights, 4. Application Unique Identifier (UID), and 5. Application Sandboxing.

Each Android app has the following core elements:

- Manifest file, which resides in the root of the app hierarchy, app metadata, components and permissions required by an app are registered herein;
- Activity – the user interface (UI) of the app is usually placed in an activity, every app must have an activity, which must be initiated by the user after installation of an app for it to be noticed by the system [9].
- Services, it is an independent entity of an application, usually used to initiate services in the background and absent of a UI. Although it is separate to other components of the app, it always runs within the context of the parent activity or the broadcast receiver which initiated it;
- Content Providers, these allow apps to share data between each other or to save information in either simple files or SQLite databases;
- Broadcast Receivers, Android OS broadcasts events e.g. Boot up completion for all apps to listen to and apps can also broadcast events for other app to listen to. Similar to a service, a Broadcast Receiver does not contain a UI.
- Intents, an intent is an abstract description of an operation to be performed.

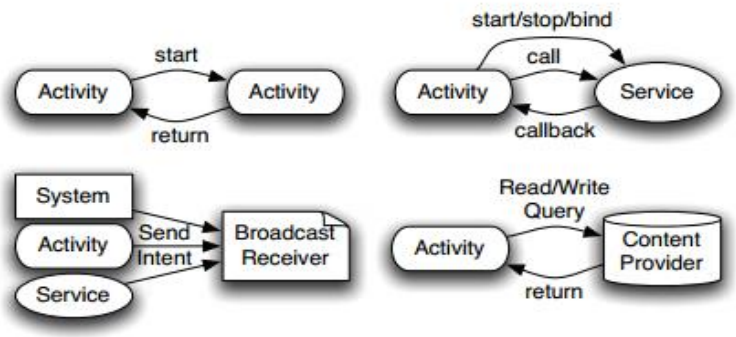

Figure 2 Android Component IPC

The above components play a crucial role in establish a secure **Inter-Process Communication (IPC)** to share data, reuse components, and provide message passing between applications/processes. The IPC is enabled via two methods, Binder and Intents; Intents provide a facility for performing late runtime binding between the code in different applications. Its most significant use is in the launching of activities, where it can be thought of as the glue between activities. On the other hand Binder is a lightweight Remote Procedure Communication (RPC) mechanism. It is a top level abstraction of Intent. [10]

Each application installed on the Android OS has the */data* folder which contains application settings and data, with the full application name as folder-title (e.g. com.mtech.projectapp). Without making changes to the OS, this folder is not accessible for other applications besides the app itself and system apps. If malware can accesses an application specific /data folder, they may change settings or content for that specific application. This is a form of sandboxing, whereby each app operates within its own context.

The Permission Model[10] [11][12][21] is the most import facet of Android Security, permission are explicitly declared in an application manifest during development and they are mainly used to restrict access levels granted to applications. Permissions are assigned to apps at installion time, but granted at runtime. This means an application can register a permission, but it might be denied it in real time, this is determined by a concept known as protection levels, which can be either 'Normal' , 'dangerous' or 'signature' or 'signatureOrSystem. These levels are checked at install time, normal permissions are always granted, dangerous permissions are granted after user confirmations of the permissions. Signature level permissions are only granted if the requesting app is signed by identical developer keys that signed the requesting package . signatureOrSystem level are used by apps which have been signed by the same key as the system image. The problem with permissions is that it is an all or nothing approach, a user must accept all permissions requested by an application at installation, and choose the ones they want.

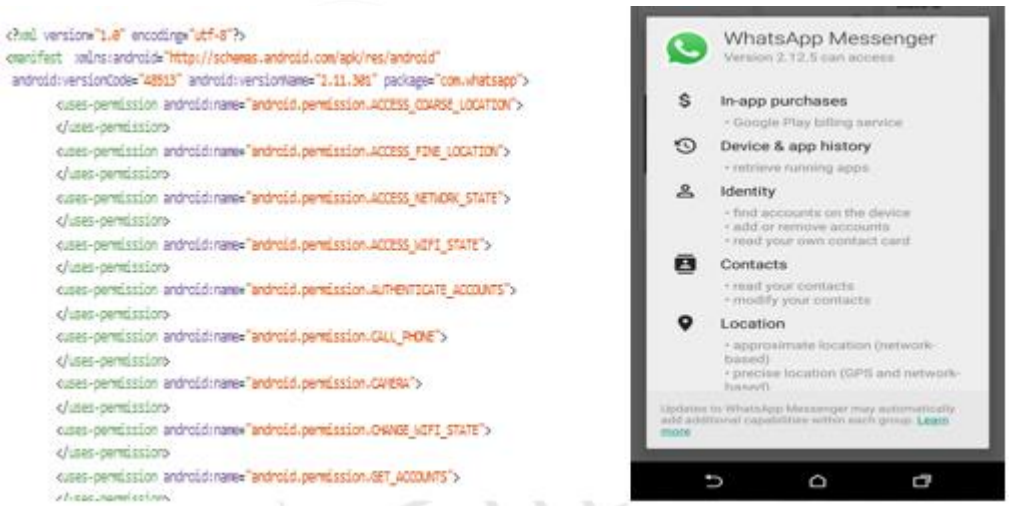

Figure 3 Permission Labels for com.whatsapp at Install time (right) & in Manifest File (left)

B. ***Android Malware***

There are various categorizations for Android malware. Jiang and Zhou [14], identified privilege *escalation, remote control, financial charges, and personal information stealing* as the most common malware type. Whilst Halilovic et. al [15], divided malware into three different groups: 1) malicious applications that cause a threat to user experience, 2) applications that generate extra costs, 3) applications that steal private information. In a much broader context, we can distinguish malware nomenclature as Trojans, Spyware, Root permission acquisition, Botnets and new variations of botnets in the form of ransomware. Ransomware is mainly used to get financial compensation from the victim.

C. Attack Vectors

These represent channels or possible ways in which malware can get onto user devices, and [14] found that the most common attack vectors are:

Repackaging - Repackaging is one of the most common techniques malware authors use to piggyback malicious payloads into popular applications. Malware authors identify legitimate popular apps, disassemble them, enclose malicious payloads, and then re-assemble and submit the new apps to official and/or alternative Android Markets.

Drive-by Download – malware authors elicit user to download malicious apps via fake adverts, phishing …etc.

Update Attack - Malware developers may still use repackaging but, instead of enclosing the affected code to the app, they includes an update component that will fetch or download the malicious payloads at runtime.

D. Malware Detection Methods

There are two predominant methods for malware detection: static and dynamic analysis of software.

1) **Static Analysis –** this method specifically looks for suspicious features at the binaries' source code. It usually involves decompilation, pattern matching and static analysis of the various features from source code. In the Android malware domain static analysis has been mainly focused on detecting malware using permissions requested, API calls, and various other features in the source code. [16].

2) **Dynamic analysis** - mainly focuses on behavior of a binary file as it executes and monitoring its behavior. [17][18]

We can further more concisely classify the above mentioned methods into the following techniques:

Signature-based detection – this is the most widely used method of detecting malicious applications. Malicious applications are identified used *signatures.* These signatures will need to be updated periodically.

Anomaly Based Detection – this monitor activities of applications and looks at deviations from normal usage patterns.

Behavior Based Detection - focused on analyzing the behavior of a program to decide if it is malicious or not. It may utilize Machine Learning (ML) approach for learning patterns and behaviors from known malware, and then to predict the unknown one.

E. Machine Learning in Malware Detection

Signature based detection suffers the drawback of not being able to detect new or metamorphic malware. It fails mainly because of encryption, polymorphism and other obfuscation methods. This has led to the use of ML in malware detection.

The key feature of Machine Learning (ML) is that it allows to build adaptable systems based on collected data. ML is traditionally defined as the "Field of study that gives computers the ability to **learn without being explicitly programmed**."[19][22]

It is possible to achieve sufficient level of detection of malware absent human interaction. For successful detection, there should be present two parts: learning algorithm and training labeled dataset with malicious and benign applications [20], after successful training we can then test unseen data. A generalized ML approach with both training and testing is shown in the Figure 4.

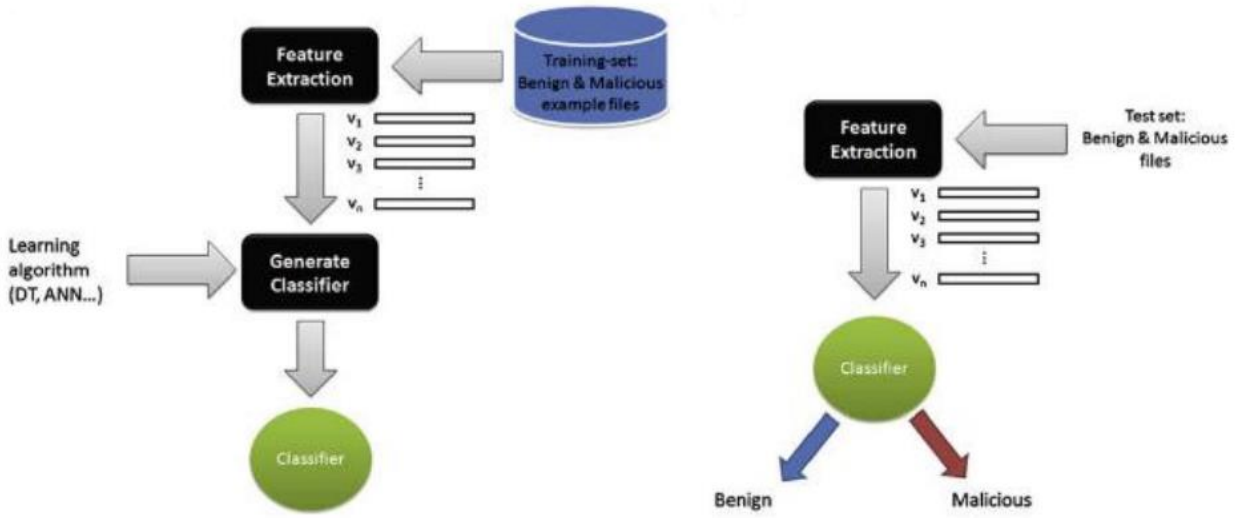

Figure 4: General Machine Learning Process

Multiple training algorithms are at our disposal when we intend to use ML for malware classification such as; Artificial Neural Networks, Bayesian Networks, Naïve Bayes, Decision Trees, Support Vector Machine(SVM), Boosted Algorithms, K-Nearest Neighbor, VFI and OneR. The next two chapters details the approach we used in our research.

III. DESIGN & METHODOLOGY

1) ***Design***

Client Side Application

Our design takes a client –server approach to detect malicious apps on the client device. On the client, a lightweight client app listens for applications installs and uploads an apk onto a remote centralized server. The user can also upload an apk to perform analysis on the submitted file. This is has problems because it results in bandwidth overhead as some apps are large. If an application has been installed already, we provide the functionality to retrieve permissions from an installed app and offload them to the remote server, to be tested using a ML model trained prior. Due to the fact that Internet connectivity will not always be available on mobile we address that concern by providing an on the cloud archiving service which stores records of classified application and we generate a unique signature for each scanned file, these signatures are small and they are regularly synchronized with the client. Therefore before a client scans or uploads a file it first queries the local database to check if it has been scanned already.

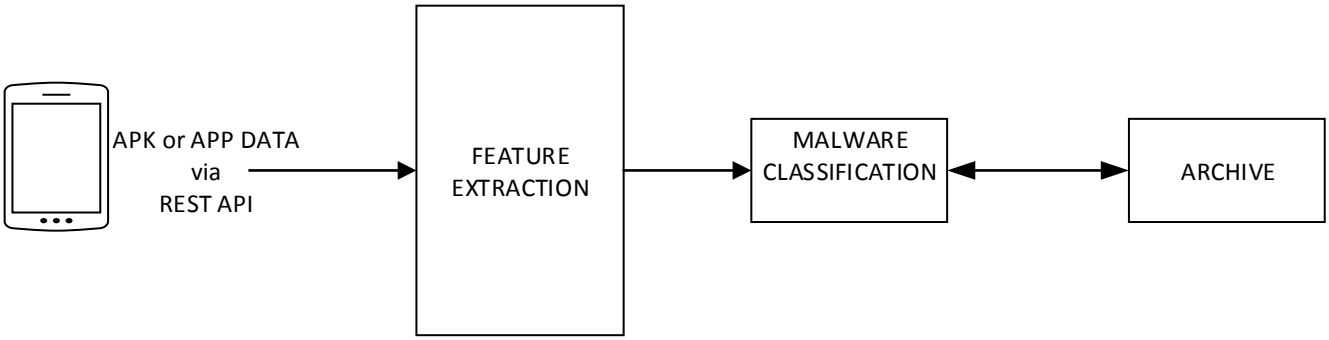

Figure 5: System Architecture

Server Side Application

The web service is a Python web service with a RESTful API and it waits for incoming HTTP requests. When the web service receives JSON data stream containing permissions it deserialises the incoming data, and sends it to a method to extract and vectorize permissions into a vector to be input into the test model. The classifier will take the input vector and will return a binary output either **0**, when an application is benign and **1** if an application is malicious, the resulting classification is saved in the archive and a signature for the application is generated. A response is then sent back to the device to alert the user of the result. The scanned app data will be used to feed and improve the training model overtime.

Feature Extraction – for feature extraction we use Androguard, a tool used for analysis of apks, reverse engineering of Android apps. We created a Python script to automatically extract permissions from uploaded apks.

```
from androguard.core.bytecodes import apk, dvm
from androguard.core.analysis import analysis
import re
def get_permissions(apk_file):
  app = apk.APK(apk_file)
  perms = app.get_permissions()
  return perms
```

Figure 6. Androguard Snippet to get a list of permissions from an apk file

The list of permissions collected from the apk or received from the client are then used to automatically create a feature vector to be input into the test model.

Feature Vector

Our feature vector is defined as follows: android has a fixed number of permissions that app developer can assign to their apps, therefore we take the set of all possible permissions and we create a binary vector of Boolean values, where a permission is requested by a specific app we set it as **1** and where it is absent we set a **0.** As shown below;

App_id [0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 ,
0 , 0 , 1 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 ,
0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 ,
0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 ,
0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 1 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 ,
0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 1 , 0 , 0 , 0 , 0 ,
0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 ,
0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 ,
0 , 0 , 0 , 0 , 1 , 0 , 0 , 0 , 0 , 0 , 1 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , malware]

Malware classification

For malware training and testing, we initially used the WEKA [23] platform because of its ease of use in rapid prototyping and testing ml methods. We later switched to Python Scikit-Learn [24] for developing the system mainly due to our familiarity with Python, automatic feature generation is handled using Pandas and Numpy. We tested multiple machine learning algorithms to build our model with. The performance of each is evaluated in the next chapter. After training our test model it is possible to load our saved model for use in the automated system.

IV. EVALUATION & RESULTS

Dataset

To train our model we used a limited dataset of 500 benign applications downloaded from the Google Play Store, as well as 300 malware gathered from various sources which include Contagio Dump, Virus share.

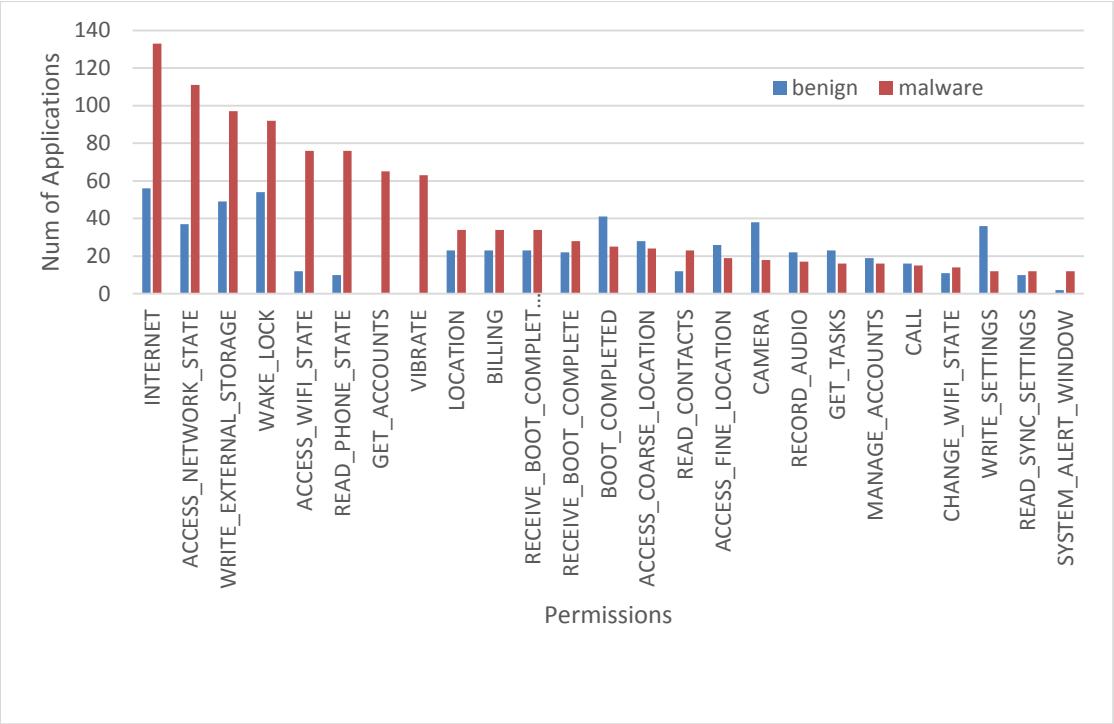

Figure 6: Top 25 requested permissions in the dataset

In Fig 6 we show the most common permissions requested by both malware and benign apps. INTERNT permission is the most frequently requested in both classes followed by WAKE_LOCK and WRITE_EXTERNAL_STORAGE.

We tested six ML algorithms namely. **Naïve Bayes (NB),** NB examines discrete variables, and draws a conclusion by calculating their probabilities. **RandomForest**, it is a decision tree based algorithm that is efficient in malware detection. **MultiLayerPerceptron** is classifier that uses back propagation to classify instances. **Simple Logistic Regression**, it is used for building linear logistic regression models. **J48,** relies on the feature values to classify instances. A decision tree consists of nodes and leave, in this case the nodes are our features and leaves are the class i.e. malware or benign. We used 2 methods to evaluate our model. Initially we trained the model and obtained the performance metrics for the trained model then, we also used 5 fold cross validation and then testing using unseen test data. Table 1, 2, 4 show the results obtained from the tests. Usually for determining the quality of the system the error rate is used as the measurement metric but in the case of a skewed class, where positive examples far outnumber negative examples, error rates are not informative. Instead we rely on precision and recall.

Precision measures of the predicted positive, what percentage is actually positive.

Precision = true positive / (true position + false positive)

Recall measures how many positive examples were correctly predicted.

Recall = true positive / (true positive + false negative).

The **F score** combines precision and recall into a single number for comparison.

F Score = 2 * P * R / (P + R)

Table 1: Training Set Results

Algorithm	**Precision (TP/TP+FP)**	**Recall(TP+FN)**	**F-Score**
NaiveBayes	0.888	0.907	0.898
RandomForest	0.99	0.99	0.99
MultiLayerPerceptron	0.986	1	0.993
J48	0.913	0.909	0.905
SimpleLogisticRegression	0.944	0.944	0.944

Table 2: 5 Fold Cross Validation

Algorithm	**Precision (TP/TP+FP)**	**Recall(TP+FN)**	**F-Score**
NaiveBayes	0.881	0.9	0.89
RandomForest	0.878	0.879	0.878
MultiLayerPerceptron	0.899	0.886	0.892
J48	0.863	0.864	0.57
SimpleLogisticRegression	0.877	0.879	0.877

Table 3: Actual Test Set of Play Store Apps

Algorithm	Precision (TP/TP+FP)	Recall(TP+FN)	F-Score
NaiveBayes	0.857	0.9	0.878
RandomForest	0.893	0.894	0.894
MultiLayerPerceptron	0.912	0.893	0.903
J48	0.867	0.867	0.864
SimpleLogisticRegression	0.893	0.894	0.891

MultiPerceptron has an excellent detection rate of 89% on the unseen data, but it is slower than the others, Second best is Random Forest.

V. DISCUSSION & CONCLUSION

We present an automated system for classification of Android malware that is capable of continuously retraining to get better results. Results show that our system is capable of achieving acceptable detection rates with adequate recall and precision. Whilst our method looks promising we wish to include other features into our detection model to ensure a better detection rate, As well as reducing noise in our classifier, by reducing the number of features used in the model and increasing the training dataset.

REFERENCES

[1] "ESet," [Online]. Available: http://www.virusradar.com/en/Android_Simplocker.A/description. [Accessed 20 JANUARY 2015].

[2] T. Project. [Online]. Available: https://www.torproject.org/. [Accessed 24 December 2014].

[3] "BlueBox Security," [Online]. Available: https://bluebox.com/technical/android-fake-id-vulnerability/. [Accessed 20 January 2015].

[4] J. Xuaxian, "An Evaluation of the Application ("App") Verification Service in Android 4.2," 2012.

[5] "Google Play Store," [Online]. Available: https://play.google.com/store. [Accessed 20 December 2014].

[6] "Contagio Dump," [Online]. Available: http://contagiominidump.blogspot.in/. [Accessed 20 December 2014].

[7] "Virus Share," [Online]. Available: http://virusshare.com/. [Accessed 20 December 2014].

[8] Google, "Android Open Source Project," [Online]. Available: https://code.google.com/p/android/. [Accessed 23 January 2014].

[9] "Android Developer," [Online]. Available: http://developer.android.com/about/versions/android-3.1.html#launchcontrols. [Accessed 2 January 2015].

[10] "Android security overview." http://source.android.com/tech/security/, [Accessed 20 January 2014].

[11] Y. F. U. K. Y. Asaf Shabtai, " Google android: A comprehensive security assessment," *IEEE Security and Privacy*, no. 8, pp. 35-44, 2010.

[12] S. Bell, "Secure Honey," [Online]. Available: http://securehoney.net/blog/how-to-dissect-android-simplelocker-ransomware.html. [Accessed 23 December 2014].

[13] "Amzon Play Store," [Online]. Available: http://www.amazon.com/gp/feature.html?ie=UTF8&docId=1001067621. [Accessed 20 December 2014]

[14] X. J. Yajin Zhou, "Dissecting Android Malware: Characterization and Evolution," in *Proceedings of the 33rd IEEE Symposium on Security and Privacy*, 2012.

[15] M. Halilovic and A. Subasi, "Intrusion Detection on Smartphones," arXiv e-print 1211.6610, Nov. 2012.

[16] H. Peng et. al, "Using probabilistic generative models for ranking risks," in In Proc. of ACM Conference on Computer and Communications Security (CCS), 2012.

[17] E. e. al, " Taintdroid: an information-flow tracking system for realtime privacy monitoring on smartphones.," in *9th USENIX conference on Operating systems design and implementation*, 2010.

[18] N. e. al, "Apex: Extending Android Permission Model," *ASIACCS*, p. 13–16, 2010.

[19] Arthur Samuel (1959).

[20] Kononenko, I. & Kukar, M. 2007. Machine Learning and Data Mining: Introduction to Principles and Algorithms.

[21] "Android permission api guides." http://developer.android.com/guide/topics/manifest/permission-element.html, December 2014

[22] "Machine Learning Coursera," [Online]. Available: https://class.coursera.org/ml [Accessed 20 December 2014]

[23] "Weka" [Online]. Available: http://www.cs.waikato.ac.nz/ml/weka/ [Accessed 20 January 2015]

[24] "Scikit-learn" [Online]. Available: http://scikit-learn.org/stable/ [Accessed 20 January 2015]