

# Malware Application Detection System For Android

Dhananjay Gandole<sup>1</sup>, Vikram Ghule<sup>2</sup>, Mahesh Garje<sup>3</sup>, Prof. P.P.Nimbalkar<sup>4</sup>

<sup>1, 2, 3, 4</sup> Department Of Computer Engineering

<sup>1, 2, 3, 4</sup> Jspm's Imperical College Of Engg & Research, Wagholi, Pune

**Abstract-** *In this paper we present Permission as well as String Based Anomaly Detection System for detecting Meaningful deviation in a mobile application's network behavior. The main goal of Proposed system is to protect mobile device users and avoid uncertainty of users. Identification of republished popular applications injected with a malicious code (i.e., repackaging). More specifically, we attempt to detect a new type of mobile malware with self-updating capabilities that were recently found on the official Google Android Marketplace.*

*Android applications are becoming increasingly because android phones are wide spread and steadily gaining popularity. Unfortunately, such phenomenon draws attention to malicious application developers so that malicious applications are increasing rapidly. These malicious applications are capable to stealing sensitive and private user information and to damage the phone hardware as well. Therefore it is highly essential to identify and detect these malicious applications which is present in both official Android Market and alternative application markets.*

**Keywords-** Android Security, Android System, Permission Usage Analysis, Malware Detection.

## I. INTRODUCTION

Nowadays Smartphone's become exploring for personal or business use. There will be an estimated 1.368 Billion of smartphones shipped globally 2015, growing 14.6% on year, according to Digitimes Research. This number increased 20% over the last year. Meantime, smartphone platforms have seen a massive surge in malwares.

As the official application (or app) market, Google's Play store provides a platform of delivering apps for Android smartphones and mobile devices. There are many third-party app markets providing similar platforms. App developers publish their apps on the Google's play or on the third-party app markets, where end users download and install their interested apps on their Android smart phones. Obviously, how detect and keep the large number of malware out of the application (or app) markets is an emerging, crucial, but challenging issue.

## II. RELATED WORK

There has been significant work on problem of detecting malware on mobile devices. Several approaches of monitor the power usage of applications, and report anomalous consumption. Others monitor system calls and attempt to detect which unusual system call patterns. Other approaches use more traditional comparison with known malwares, or other heuristics. The more general field of malware detection is host to the wider range of approaches. In Sahs et al.[1] traditional static analysis of approaches such as, which focus on comparing programs to known malware based on the program code, looking for the signatures or using other heuristics. Other approaches are focus on using machine learning and data mining approaches for malware detection. Train a neural network to detect boot sector viruses, based on byte string trigrams. Compare three machine learning algorithms trained on three features: DLL and system calls made by the program, strings found in the program binary, and a raw hexadecimal representation of the binary.

In Cen et al. [3] the paradigm for program distribution on these mobile devices also differs from that of the traditional PCs. Many developers are releasing applications to one or a few central application markets. While there are third party application stores, currently all popular mobile device platforms have central application stores as the primary mechanism of application distribution.

Crowdroid is a machine learning-based framework that recognizes Trojan-like malware on Android smartphones, by analyzing the number of times each system call has been issued by an application during the execution of an action that requires user interaction. A genuine application differs from its trojanized version, since it issues different types and a different number of system calls. Crowdroid builds a vector of m features (the Android system calls). Another IDS that relies on machine learning techniques is Andromaly which monitors both the smartphone and user's behaviors by observing several parameters, spanning from sensors activities to CPU usage. 88 features are used to describe these behaviors; the features are then pre-processed by feature selection algorithms. The authors developed four malicious applications to evaluate the ability to detect anomalies. MADAM: a Multi-Level Anomaly Detector for Android Malware uses 13 features to detect android malware for both kernel level and user level. MADAM has been tested on real malware found in the wild and uses a global-monitoring approach that is able to detect

malware contained in unknown applications, i.e. not previously classified Aung et al. [4].

### III. MALWARE CHARACTERIZATION

In this section, we present a systematic characterization of existing Android malware, ranging from their installation, activation, to the carried malicious payloads.

#### A. Malware Installation

By manually analyzing malware samples in our collection, we categorize existing ways Android malware use to install onto user phones and generalize them into three main social engineering-based techniques, i.e., repackaging, update attack, and drive-by download. These techniques are not mutually exclusive as different variants of the same type may use different techniques to entice users for downloading.

##### 1) Repackaging

Repackaging is one of the most common techniques malware authors use to piggyback malicious payloads into popular applications (or simply apps). In essence, malware authors may locate and download popular apps, disassemble them, enclose malicious payloads, and then re-assemble and submit the new apps to official and/or alternative Android Markets. Users could be vulnerable by being enticed to download and install these infected apps. To quantify the use of repackaging technique among our collection, we take the following approach: if a sample shares the same package name with an app in the official Android Market, we then download the official app (if free) and manually compare the difference, which typically contains the malicious payload added by malware authors. If the original app is not available, we choose to disassemble the malware sample and manually determine whether the malicious payload is a natural part of the main functionality of the host app. If not, it is considered as repackaged app.

##### 2) Update Attack

The first technique typically piggybacks the entire malicious payloads into host apps, which could potentially expose their presence. The second technique makes it difficult for detection. Specifically, it may still repackage popular apps. But instead of enclosing the payload as a whole, it only includes an update component that will fetch or download the malicious payloads at runtime. As a result, a static scanning of host apps may fail to capture the malicious payloads. In our dataset, there are four malware families, i.e., BaseBridge,

DroidKungFuUpdate, AnserverBot, and Plankton, that adopt this attack.

### IV. PROPOSED SYSTEM

A system architecture or systems architecture is the conceptual model that defines the structure, behavior, and more views of a system. An architecture description is a formal description and representation of a system, organized in a way that supports reasoning about the structures of the system.

The overall architecture diagram is given below. Our system begins with the phase of downloading Android applications. Both genuine and malware applications need to be downloaded to train and test our system. Malware Android applications cannot be easily downloaded from the internet. We became a member in virusshare.com- a malware repository used for research and analysis purposes. After we have created a Repository of Android applications, we need to extract the AndroidManifest.xml and classes.dex \_les from each of the .apk package.

Both the \_les will be in encrypted form. In order to decrypt the manifest xml \_le, we use AXMLPrinter2.jar tool. To decompile the dex \_le (compiled java source code), we use dex2jar.jar tool. Both the tools are automated and executed using Windows PowerShell. Since we need to perform the decrypting of xml \_les and decompiling of dex \_les for all the .apk packages we have downloaded, it is literally impossible to separately use the tool for every application.

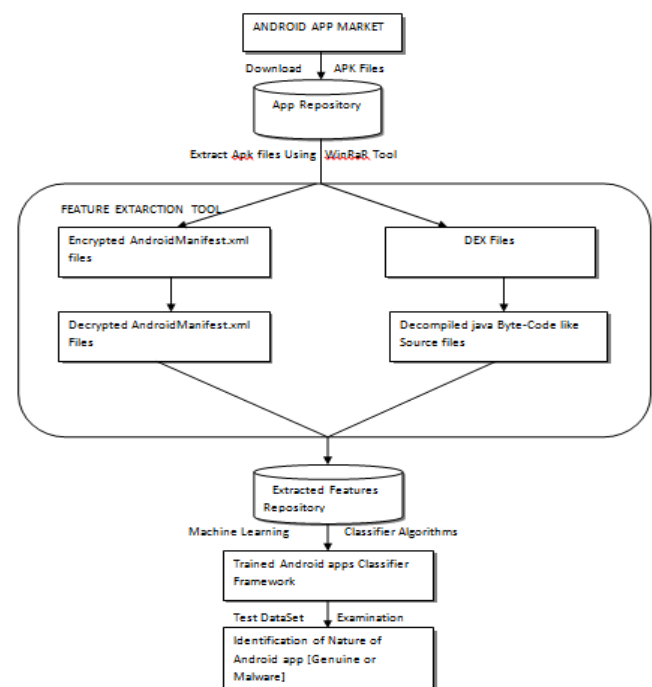


Fig. Architecture Diagram

**V. ALGORITHM**

In this section we first use J48 algorithm for generate a decision tree and Second algorithm use Naïve Bayes for finding the Probability of the Dataset.

**1) J48 (OR) C4.5 :**

C4.5 is an algorithm used to generate a decision tree developed by Ross Quinlan. C4.5 is an extension of Quinlan's earlier ID3 algorithm. The decision trees generated by C4.5 can be used for classification, and for this reason, C4.5 is often referred to as a statistical classifier.

**2) NAÏVE BAYES :**

In machine learning, naive Bayes classifiers are a family of simple probabilistic classifiers based on applying Bayes' theorem with strong (naive) independence assumptions between the features.

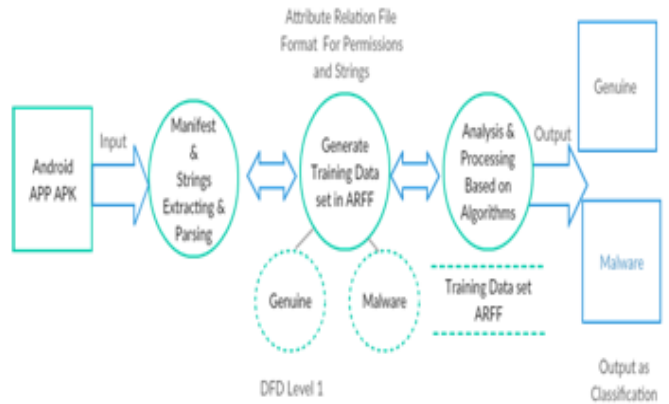
Naive Bayes has been studied extensively since the 1950s. It was introduced under a different name into the text retrieval community in the early 1960s, and remains a popular (baseline) method for text categorization, the problem of judging documents as belonging to one category or the other (such as spam or legitimate, sports or politics, etc.) with word frequencies as the features. With appropriate pre-processing, it is competitive in this domain with more advanced methods including support vector machines. It also finds application in automatic medical diagnosis. Naive Bayes classifiers are highly scalable, requiring a number of parameters linear in the number of variables (features/predictors) in a learning problem. Maximum-likelihood training can be done by evaluating a closed-form expression, which takes linear time, rather than by expensive iterative approximation as used for many other types of classifiers.

**DATAFLOW DIAGRAM**

**LEVEL 0:**

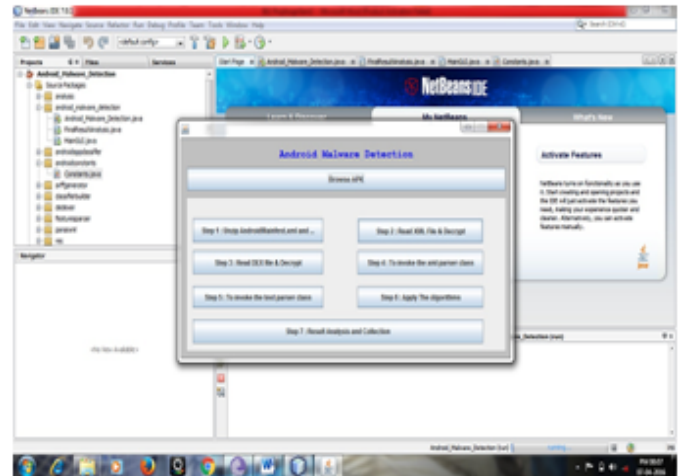


**LEVEL 1:**

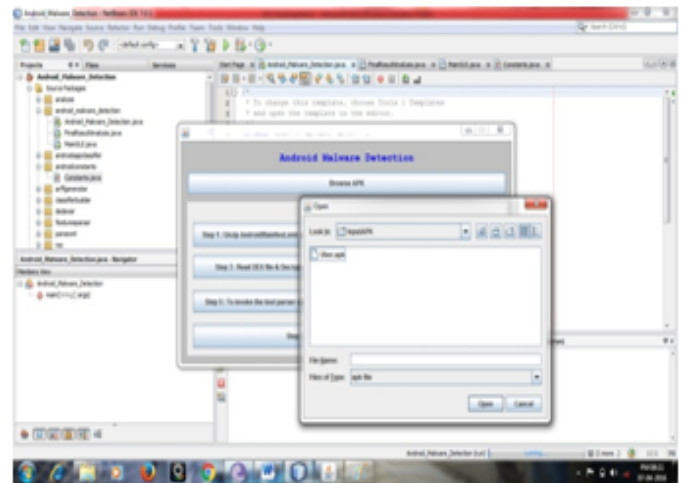


**VI. RESULTS**

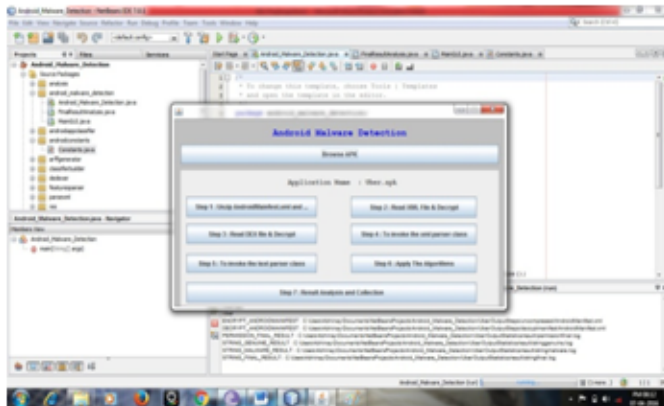
**1. After running obtained the GUI of project.**



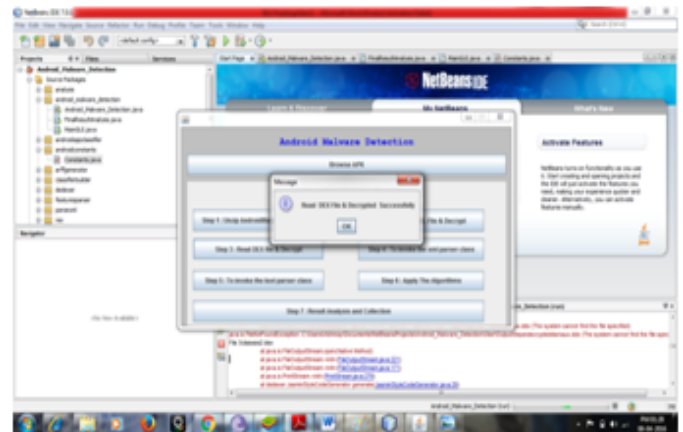
**2. Browsing the .apk from directory.**



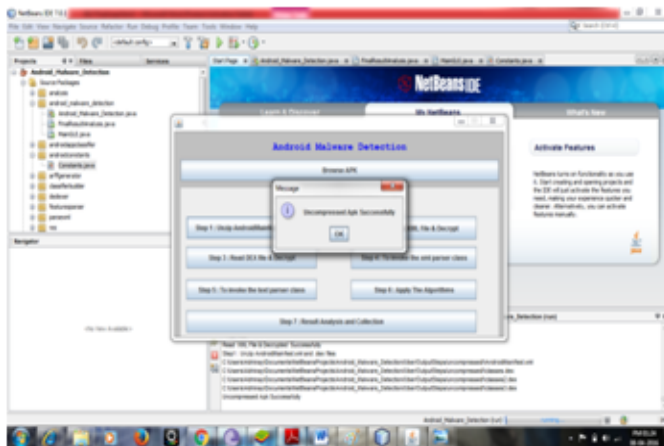
3. When .apk file successful browse.



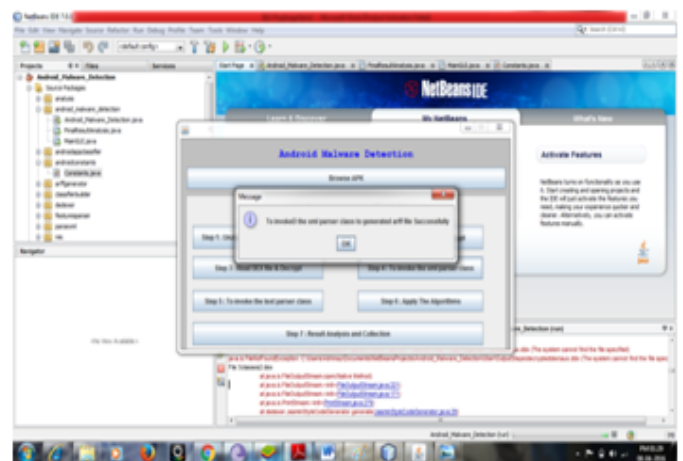
6. Read the classes.dex files.



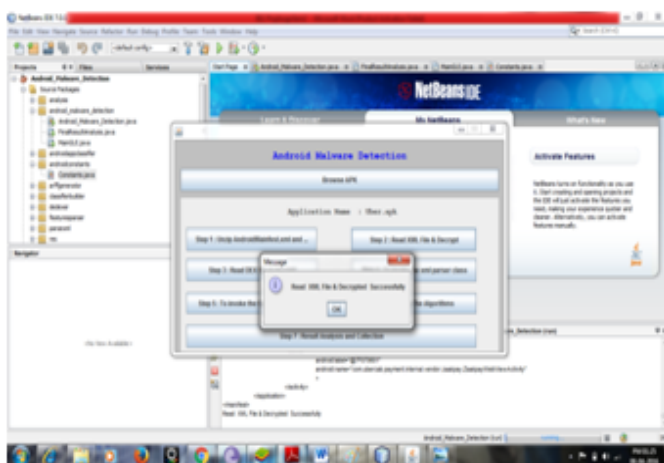
4. When unzip .apk file successfully.



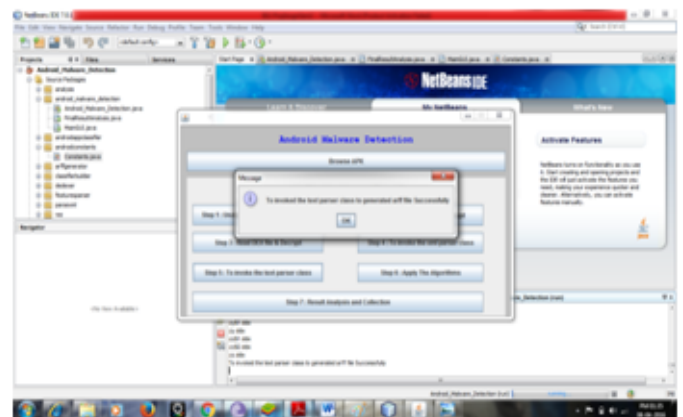
7. Parsing the AndroidManifest.xml file and generated ARFFOnPermission file.



5. Reading the AndroidManifest.xml file.

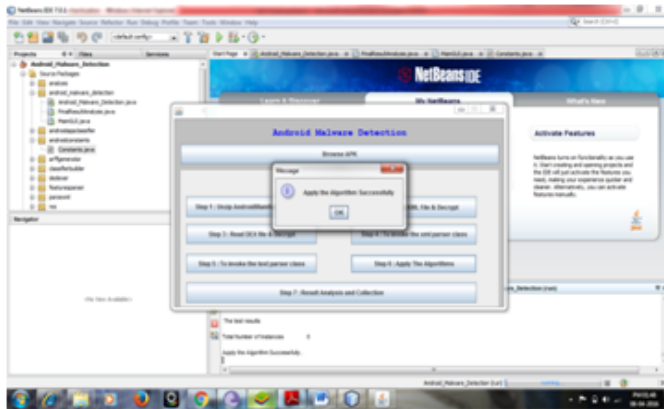


8. Parsing the classes.dex files and generated ARFFOnStrings file.

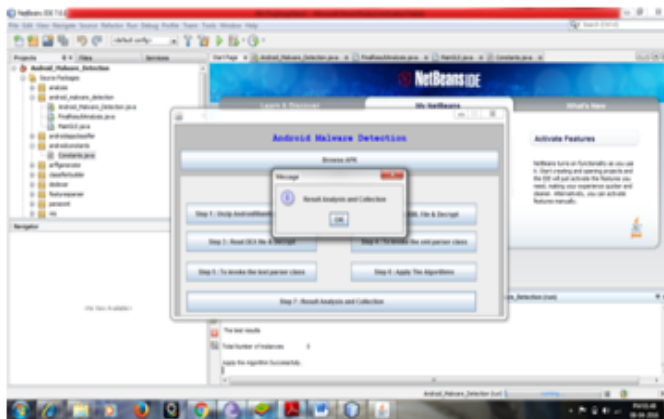




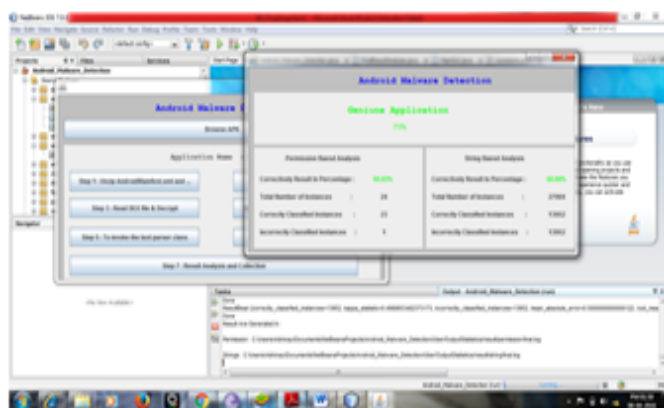
9. Applying the algorithms.



10. After apply algorithms result get analysis and collected.



11. Final Result.



VII. CONCLUSION

We had survey on framework for classifying Android applications whether they are malware or normal applications. To generate the models, we have extracted several permission features from several downloaded applications from android markets.

ACKNOWLEDGMENT

We would like to thank our guide and various technological experts who researches about malware detection and improve the result by implementing new methods. We would also like to thank Google for providing details on different issues on malware detection and about other related techniques.

REFERENCES

- [1] “A Machine Learning Approach to Android Malware Detection”, Justin Sahs and Latifur Khan {2012}.
- [2] “Exploring Permissions-induced Risk in Android Applications for Malicious Application Detection”, Wei Wang, Xing Wang, Dawei Feng, Jiqiang Liu, Zhen Han, Xiangliang Zhang {2014}.
- [3] “A Probabilistic Discriminative Model for Android Malware Detection with Decompiled Source Code”, Lei Cen, Christoher S. Gates, Luo Si, and Ninghui Li. - 2015)
- [4] ” Permission-Based Android Malware Detection”, Zarni Aung, Win Zaw – 2013
- [5] Dong-uk Kim, Jeongtae Kim, Sehun Kim, "Malicious Application Detection Framework using Feature Extraction Tool on Android Market".
- [6] BorjaSanz, Igor Santos, Carlos Laorden, XabierUgarte-Pedrero and Pablo Garcia Bringas, “MADS: Malicious Android Applications Detection through String Analysis”.
- [7] “Mobile malware detection through analysis of deviations in application network behaviour”, A. Shabtai, L. Tenenboim-Chekina, D. Mimran, L. Rokach, B. Shapira, Y. Elovici.
- [8] DREBIN: Effective and Explainable Detection of Android Malware in Your Pocket” ,Daniel Arp, Michael Spreitzenbarth, Malte H’ubner, Hugo Gascon, Konrad Rieck.
- [9] “Dissecting Android Malware: Characterization and Evolution”, Yajin Zhou, Xuxian Jiang.
- [10] “Permlyzer: Analyzing Permission Usage in Android Applications”, Wei Xu, Fangfang Zhang, and Sencun Zhu.