Design Engineering

ISSN: 0011-9342 | Year 2021
Issue: 6 | Pages: 7749- 7766

A Novel Portable Executable Malware Detection Using Random Forest With Feature Set Generation Algorithm (Rf-Fsga)

[1]Mrs. M .Meena Krithika, [2]Dr. E. Ramadevi

[1]Assistant professor, [2]Assistant professor,
[1]Department of computer science, [2]Department of computer applications,
[1,2]NGM college, Pollachi, TamilNadu , India.

ABSTRACT –The open source nature of Android Operating System has pulled in more extensive appropriation of the system by various types of developers. This wonder has additionally cultivated a dramatic expansion of gadgets running the Android OS into various areas of the economy. A coordinated list of capabilities has been amalgamated as a mix of decreased executable header field's crude worth and construed values. In this phase, propose a a novel portable executable malware detection using random forest with feature set generation algorithm (RF-FSGA) for malicious PE file detection, in like manner shows improvement in accuracy by utilizing derived features related to a subset of existing raw features over the accuracy of simply raw features. In the experiments directed on the novel test informational collection the accuracy was seen as 89:23% for the integrated feature set which is 15% enhancement for accuracy accomplished with raw-feature set alone.

Keywords: [Android, Operating System, Malware, Machine,Portable executable File.]

1. INTRODUCTION

Malicious program or malware is a purposefully composed program to enjoy different malicious exercises, going from client's information stealing to cyber espionage. The conduct dynamism uncovered by the malware is dependent on different factors, for example, nature of the attack, sophisticated technology and the fast expansion in exploitable vulnerabilities. Malware attacks likewise expanded alongside the fast growth in the utilization of computerized devices and internet. The remarkable expansion in the creation of new malware over the most recent five years made the malware detection as a difficult research issue 1. Malware detection is the technique for recognizing malware in the end devices or organizations. Signature based detection and Non-signature based detection are the two significant classes of malware detection strategies used today.

A new report has demonstrated that there are around 700,000 Android Apps right now accessible on the market. This notoriety of the Android system has prompted a colossal expansion in the spreading of Android malware. These malware are principally dispersed in markets operated by outsiders, however even the Google Android Market can't ensure that the entirety of its recorded applications are sans threat. The threats for Android incorporate Phishing, Banking-Trojans, Spyware, Bots, Root Exploits, SMS Fraud, Premium Dialers and Fake Installers. There have additionally been reports about Download-Trojans Apps that download their malicious code after installation which implies that these Apps can't be handily detected by Google's technology during publication in the Google Android Market.

In outline, malware applications ordinarily utilize following three types of penetration techniques for installation, activation, and running on the Android system:

Repackaging is perhaps the most well-known techniques for malware developers to install malicious applications on an Android platform. These types of approaches typically start from famous legitimate Apps and abuse them as malware. The developers typically download famous Apps, dismantle them, add their own malicious codes, and afterward re-gather and upload the new App to official or elective markets.

Updating technique is harder for detection. Malware developer may in any case utilize repackaging yet as opposed to encasing the inflict code to the App; they incorporate an update segment that will download malicious code at runtime.

Downloading is the traditional attack technique, malware developer need enticing users to download fascinating and attractive Apps.

The amount of Android malware is growing quickly; especially, progressively more malicious software use obfuscation technology. Traditional detection strategies for manual analysis and signature coordinating have uncovered a few problems, for example, slow detection speed and low accuracy. Lately, numerous researchers have tackled the problems of Android malware detection utilizing machine learning algorithms and had a great deal of research results. With the ascent of deep learning and the improvement of computer computing power, an ever increasing number of researchers started to utilize deep learning models to detect Android malware. +is paper proposes an Android malware detection model dependent on a hybrid deep learning model with deep conviction organization (DBN) and gate recurrent unit (GRU).

Inspired by the above observations, propose a framework for analyzing and ordering Android applications dependent on machine learning techniques.

ISSN: 0011-9342 | Year 2021
Issue: 6 | Pages: 7749- 7766

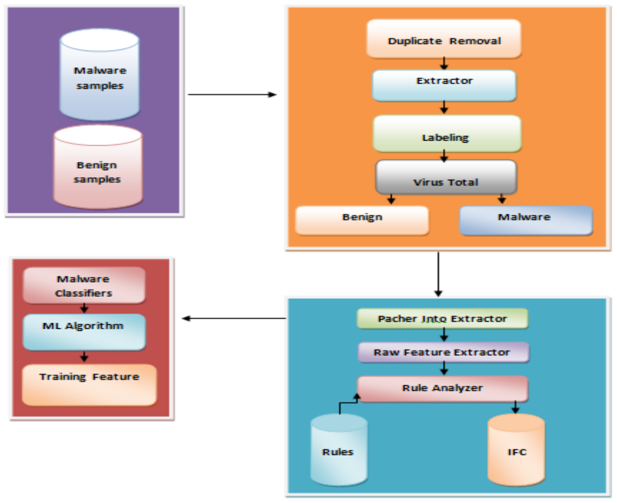


Figure 1.Block Diagram of overall work flow

The framework lies on a mix of mentioned permission and static API call behaviors, and concentrates features from these behaviors and fabricates classifiers to distinguish malicious applications. The procure 96.39% accuracy which uncovered a dependable technique to protect against malicious destructive exercises.

1.1 PE (Portable executable) file format

PE file format was developed by Microsoft and it was presented with Windows NT 3.1 in the year 1993. It is metadata of executable, dlls and object files. Since its presentation, it has been widely by researchers in different domains yet PE has found its prime importance in Malware detection systems.PE file format based approaches are effective for scanning executable files yet PE cannot be generated for android app, to classify an android app as malware or legit, different techniques are utilized. In apk file is utilized to acquire manifest and different features are separated from it to train the model the ideally chosen feature subset is gotten from the genetic algorithm at that point utilized it for further processing. In .dex files got were changed over to gray scale image and afterward feed to different models

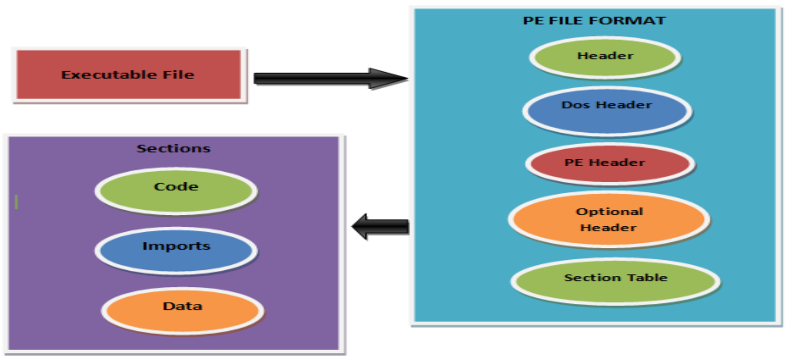


Figure 2.PE File Format

Column Name	Description
Pe.has_nx	True if the PE has NX bit set.
Pe.has_tls	True if the PE is using TLS
Pe.has_rich_header	True is a rich header is present
Pe.has_debug	True if the PE has debug section
Pe.has_exports	True if the PE has any exported symbol

Table 1: Description of second data set

1.2 Android Malware Attack Trend

Information Extraction:The malware in this classification bargains a gadget and afterward steals personal information, for example, IMEI number, client's personal information and some more.

Automatic Calls and SMS:This gathering of malware builds a client's phone bill by placing automatic calls and sending SMS to some premium numbers.

Root Exploits:These set of malware look to acquire system root advantages to assume responsibility for the system and modify the system's configuration and other system information.

Search Engine Optimizations:The malware here artificially looks for a term and re-orders taps on centered websites to grow the income of a web index or augmentation the traffic on a webpage.

Dynamically Downloaded Code:This procedure empowers an acquainted kind application with download a malicious code and passes on it in the cell phones without the customer remaining alarm.

2. EXISTING SYSTEM

2.1 Idrees et al. (2017) et.al proposed PIndroid, which was a novel Android malware applications recognition structure that utilizes authorizations and purposes highlights for the planning of models and further utilized classifier mix strategy to consolidate the classifiers for an improved show. The creators utilized 1745 application tests to lead the investigations beginning with an assessment of execution between six classifiers: Multi-Lateral Perceptron (MLP), Decision Table, Decision Tree, Random Forest, Naïve Bayesian and Support Vector Machine (SVM) using Sequential Minimal Optimization (SMO). They consolidated the Decision Table, MLP, and Decision Tree classifiers utilizing three blend plans which are Average of Probabilities, Product of Probabilities, and Majority Voting. The structure gave 99.8% True Positive location accuracy rate, 1.1% False Positive recognition rate and a F-extent of 99.7%through the Product of probability blend technique.

2.2 Adebayo (2017) et.al proposed zeroed in his work on building a strong malware identification structure by improving Apriori count utilizing atom swarm advancement and assentbased features of Android applications to improve the classification framework and detection pace of malicious applications.He used the molecule swarm optimization (PSO) to create up-and-comers (banner transporters) from the future set of Android applications for the improvement of Apriori algorithm and Apriority Association Rule (AAR).Afterward, the Author formulated rules from the generated competitors utilizing the AAR. The detection model was created by using the banner conveyors and rules to prepare seven indisputable characterization algorithms; Bayesian Classifier, Naive Bayes (NB), PART, Decision Tree (J48), Random Forest (RF), Neural Network (NN), and Classification-based Multiple Association Rule (CMAR). The absolute application information utilized was an example of 1500. The model developed from AAR-PSO performed better, having the best most elevated True Positive malware detection (TPR), lowest False positive (FPR), most elevated accuracy, and lowest blunder rate than any single model developed utilizing the individual classification algorithm.

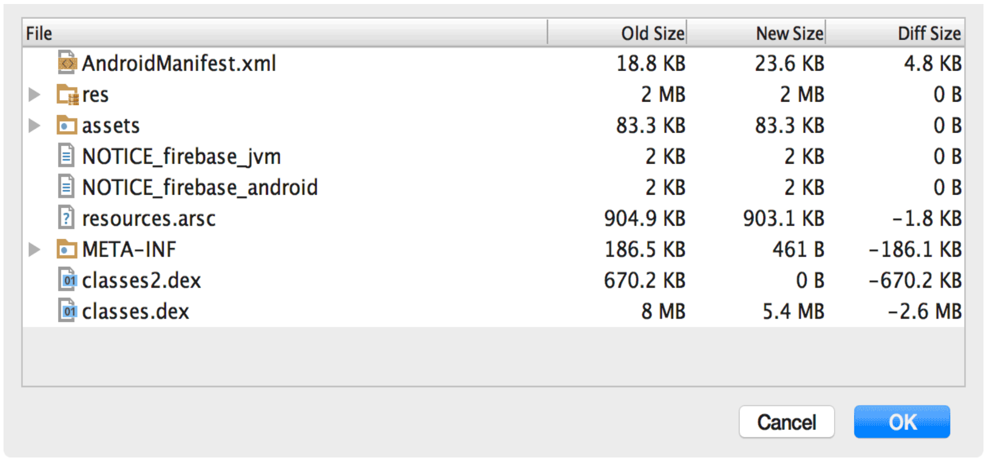
3. PROPOSED SYSTEM:

The features extraction technique consolidating dynamic analysis and static analysis is embraced. The static features are acquired by decompiling the APK file, including resource features and semantic features. The static features generate a double feature vector through one-hot encoding. The dynamic highlights are acquired by checking the related API calls during the APK running cycle. For the dynamic features related with the time arrangement, the element installing technique is utilized to generate feature vectors. In this part, we propose a feature-based a novel portable executable malware detection using random forest with feature set generation algorithm (RF-FSGA) for malicious PE file detection, in like manner shows improvement in accuracy by utilizing derived features related to a subset of existing raw features over the accuracy of simply raw features. The framework considers the application features, including permissions and API calls, to characterize Android applications behaviors. The proposed system comprises of four significant parts. The main segment is an App analyzer that decompresses the APK document of an App and concentrates AndroidManifest.xmland class files, which are essential for characterizing Apps. The subsequent component means to characterize each App based on its mentioned permissions and API calls. The third component, feature generator, completes feature.

3.1 Android Application Anatomy

All Android applications are made and arranged as Android Package (APK) file. The APK file is just a ZIP format document file that is renamed to have apk extension. The substance comprises of a Dalvik executables, resources, native libraries and a manifest file; and is generally endorsed by the developer of the application utilizing self-stamped authentication. Specifically, the APK record would ordinarily contain two envelopes (META-INF and res) and three documents (Classes.dex, AndroidManifest.xml, and Resources.arsc). The classes.dex and

AndroidManifest.xml are the most sensitive and significant components of the APK file which are normally the high targets of malware makers.



File	Old Size	New Size	Diff Size
AndroidManifest.xml	18.8 KB	23.6 KB	4.8 KB
res	2 MB	2 MB	0 B
assets	83.3 KB	83.3 KB	0 B
NOTICE_firebase_jvm	2 KB	2 KB	0 B
NOTICE_firebase_android	2 KB	2 KB	0 B
resources.arsc	904.9 KB	903.1 KB	-1.8 KB
META-INF	186.5 KB	461 B	-186.1 KB
classes2.dex	670.2 KB	0 B	-670.2 KB
classes.dex	8 MB	5.4 MB	-2.6 MB

Figure 3.Android APK file displaying its contents

The classes.dex is the dalvik Virtual Machine (VM) executable document which contains the essential working code of the application. That is, the payloads of an application are made and characterized in classes.dex record. The application code is masterminded and taken care of in dex design. The AndroidManifest.xmlgives semanticrich information about the application which incorporates the rendition and required permissions administering an app's operations.

3.2 Static Analysis Module

Static analysis and dynamic analysis are utilized to extract features. When discussing the static analysis, we actualize a decoder based on the Androguard, instrument which is one of the biggest open source projects for Android static analysis. After decompiling, we gather different features from AndroidManifest.xml, classes. dex. Table 1 shows the part of the static features we extract from a malicious sample.

Category	Value
Permission	Android. Permission. RECEIVE_SMS Android. Permission. SEND_SMS…
API	Android/telephony/Telephony Manager; get Device ID Android/telephony/Telephony Manager; get Subscribe rid..
Uses-feature	Android. Hardware.touchscreen Android.Hardware.Camer..
Application	Com.air push.Android.Message Receiver Com.air push. Android.Delivery Receiver..

Table 2.Features extracted from the APK

As is appeared in Table 2, we pick permission, API, utilizes feature, application, aim as the static features. Research finds that permission system in Android is one of the main security mechanisms; malicious software will in general demand sensitive permissions more than kind software, for example, android. Permission. SEND_SMS, and so on Likewise, utilizes feature defines the admittance to the hardware, as is appeared in Table 1, this malicious application applies admittance to the touchscreen and the camera, mentioning admittance to specific hardware often reflects harmful behavior. Application comprises of four different types of components in an application; the names of these components may help identify the famous components of malware. Expectation can be utilized to trigger malicious exercises, in this manner it is additionally indispensable to gather the plans recorded in the manifest. Dubious API calls offer admittance to sensitive data which can prompt malicious behavior. Therefore, we pick these characteristics as the static features.

APIs are similarly regularly used as key highlights in detecting Android malware. Programming interface and consent-based grouping framework were created as YARA Rule, and the API, class, and public procedures for every application are removed from AndroidManifest.xml, classes.dex and composed with YARA Rule. The proposed a list of capabilities containing the authorizations and API calls for Android malware static identification, and classifiers that used the proposed include set outperform those just with the permissions.

Dynamic Analysis

As progressively more Android malware avoid static detection through procedures, for instance, repackaging and code obfuscation, dynamicanalysis strategies based on behavioural characteristics can tackle this difficult well. Dynamic examination alludes to observing the conduct of Android application software when it is executed. The monitoring scope of most dynamic analysis techniques is predominantly admittance to sensitive data and API calls, and so forth.

Malware endeavors to evade detection by mimicking security-sensitive behaviors of amiable apps and stifling their payload to decrease the opportunity of being noticed.

3.3 Feature Sets

i. Raw Feature Set

Raw feature set is made by extracting values from all fields of three primary headers (DOS header, File Header and Optional header, including standard and Windows-specific fields) present in each PE file. Altogether, raw feature set has 55 features in which 19 features are from DOS header, 7 from File Header and rest 29 are taken from Optional header. During training e res and e res2 fields are taken out from raw feature set as they are held (as per the archive) and have no values for the samples (counting malware and kindhearted samples). Along these lines, the final raw feature set has 53 features utilized for training and testing. PE record processing and field's worth extraction are performed on Linux machine with the help of prefile python module. Bits of knowledge concerning dataset and extraction. Eq1 shows the features vector forraw feature set where RF speaks to raw feature set and DH, FH and OH speak to the header's fields of DOS header, File header and Optional header respectively.

$$\mathbf{RF} = \begin{bmatrix} \mathbf{DH} = \{\mathbf{DH_1}, \dots, \mathbf{DH_{19}}\}\mathbf{U} \\ \mathbf{FH} = \{\mathbf{FH_1}, \dots, \mathbf{FH_7}\}\mathbf{U} \\ \mathbf{OH} = \{\mathbf{OH_1}, \dots, \mathbf{OH_{29}}\}\mathbf{U} \end{bmatrix} \quad \text{----- (1)}$$

ii. Integrated feature set

The integrated feature set is made by joining a few chose raw features and a set of derived features. The proposed technique is intended to use the combinatorial benefits of both the raw and derived features as integrated features.

The proposed integrated feature set is made with the supposition that incorporating raw and derived features will improve the detection accuracy.

$$\mathbf{INF} = \begin{bmatrix} \mathbf{Raw}=\{\{\mathbf{DH_1},\dots,\mathbf{DH_6}\}\mathbf{U}\{\mathbf{FH_1}\}\mathbf{U}\{\mathbf{OH_1},\dots,\mathbf{OH_{21}}\}\}\mathbf{U} \\ \mathbf{ExpandedRaw}=\{\{\mathbf{C_1},\dots,\mathbf{C_{11}}\}\mathbf{U}\ \{\mathbf{DC_1},\dots,\mathbf{DC_{15}}\}\mathbf{U} \\ \mathbf{Derived}=\{\mathbf{D_1},\dots,\mathbf{D_{14}}\} \end{bmatrix} \text{-- (2)}$$

iii. Derived features

Derived features will be features that are derived from the raw value of PE header by approving with a set of rules. The consequence of this process is taken as the feature value. Decision for set of derived features is made upon the guidelines given for PE headers' field 3. Those fields which need to have a value from a pre\defined set of values as per the guidelines are picked as derived features.

iv. Entropy

Entropy can be defined as measure of efficiency of information stockpiling (Shannon, 1948). It is straightforwardly identified with the packing of the file, and pressed file will have high entropy bringing about high eciency of information stockpiling Yonts (2012). The pefile module has one strategy named get entropy () that calculate entropy of a given section data.

3.4 Feature Generation

Feature generation is a significant assignment of the proposed work, algorithm 1 for that sets out the significant advances performed to generate the raw and integrated feature set. All marked malware and kindhearted samples alongside crafted header rules are input to the algorithm. Two loops, one for generous and other for malware samples hurry to bring sample one by one for processing. By calling FetchFieldsValue () procedure, all values are extracted from different fields of PE sample file.

Raw feature set (for example RF) is refreshed with all extracted values and appending class name. For integrated feature set (i.e INF), Check Rules() procedure is called by passing extracted values (for example Raw Value) and rules and it returned derived values which is utilized to refresh integrated feature set alongside File Value and class name. This Check Rules() procedure accepts all raw values as input yet rules are checked against just chose fields and other field values are returned with no change. ExtractFileFeatures() procedure accepts a sample as input and return other derived values, for example, file size, file entropy identified with file properties.

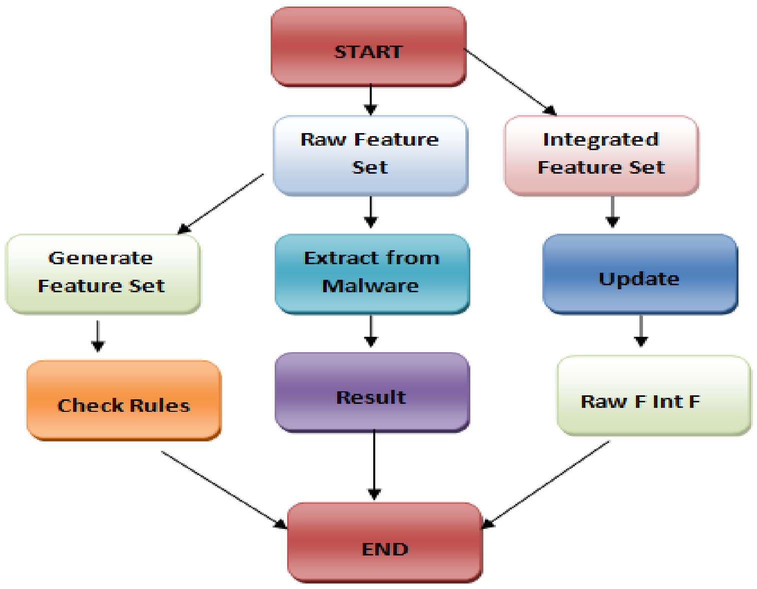


Figure 3.Flow Diagram of Proposed Modal

Static Feature Extraction

Resource features refer to features extracted from resource files put away in APK. The fundamental reason for extracting resource features is the inconsistent structure and inconsistent logic of the APK. Inconsistent structure alludes to the antiques abandoned by disguising malicious segments, achieving an irregular structure of the APK document. Inconsistent rationale alludes to the way that when malicious software is repackaged as a chivalrous application, it by and large leaves follows. The types and quantity of resource features are appeared in Table 1. A sum of 124 resource features was extracted.

Dynamic Features Extraction

Dynamic highlights are the conduct qualities of the Android application when it runs, for instance, data encryption and decryption, file perusing and composing, network data transmission, call, SMS, geographic location, and admittance to sensitive information. These behaviors can speak to the application's functions and intentions. A sum of 48 dynamic features is extracted. The extraction of these dynamic features is generally founded on observing related

API work calls. Each dynamic feature relates to a few API functions, and the all-out number of API functions is 141. A portion of the dynamic features and comparing API models are recorded.

3.5 Feature set Generation Algorithm

Feature set Generation Algorithm

```
Step 1: Strategy GENERATE FEATURE SET ($\mathfrak{M}, \mathfrak{B}, \mathfrak{R}$}
Step 2: α implies count ($\mathfrak{M}$) and β implies count ($\mathfrak{B}$)   // $\mathfrak{M}$ − malware set, $\mathfrak{B}$ − beningn set
Step 3: initialize RF [α + β][ ] is zero and INF [α + β][ ] is zero
  // RF = raw feature set
  // INF = integrated feature set
Step 4: For υ ε $\mathfrak{M}$ do                    // Extract features from malware
Step 5:     set class is zero
Step 6:     set Raw value is fetch fields value (υ) and File value is Extract file feature(υ)
Step 7:     For Value ∈ Raw Value do
Step 8:         set DerivedValue is CheckRules(Value, $\mathfrak{R}$)      // $\mathfrak{R}$ = PE field rules
Step 9:     assign RF ⟵ update RF(RawValue ∪ Class) and INF ⟵ update
            INF(FileValue ∪
            DerivedValue ∪ Class)
Step 10: For υ ε $\mathfrak{B}$ do                    // Extract features from malware
Step 11:    set class is one
Step 12:    set Raw value is fetch fields value (υ) and File value is Extract file feature(υ)
Step 13:    For Value ∈ Raw Value do
Step 14:        set DerivedValue is CheckRules(Value, $\mathfrak{R}$)     // $\mathfrak{R}$ = PE field rules
Step 15:    assign RF ⟵ update RF(RawValue ∪ Class) and INF ⟵ update
            INF(FileValue ∪
            DerivedValue ∪ Class)
Step 16: return(RF, INF)
```

The performance of the proposed integrated feature set is additionally contrasted and the PE headers' fields' values alongside DLL and API calls. The accuracy of the integrated feature set is just 0:5% not exactly the Bai et al. (2014) work, and the likely explanation may be the utilization of DLL and API calls. Commitments of the proposed work are as recorded below:

An integrated feature set is made, which has raw and derived features, based on different header fields' values of PE file. The proposed integrated feature set improves the classification accuracy of machine learning classifiers.

The proposed work additionally gives an exact proof to ease of use of different headers fields' value of PE file as useful discriminative features. Experiments' outcome plainly proposes that header fields' raw values constantly derived from these can be utilized to classify malware programs from kind-hearted programs.

It helps in contrasting the performance of different models on numerous measurements, for example, accuracy, review, exactness and f1-measure.

Dataset

Microsoft delivered approximately half terabyte for kaggle Microsoft Malware Classification Challenge containing malware (21653 get together codes). In the malware dataset, we have found that greatest size of get together code is 147.0 MB, so all the benign gathering over the 147.0 MB are not considered for the analysis. From prior assessments, we found that there are 1808 unique opcodes so in our methodology; there are 1808 features for machine learning. By then the frequency of each opcode in each malware and the benevolent file is calculated.After that in each malware and considerate file absolute opcodes weight is calculated. By then it is seen that there are 91.3 % malware file and 66 % benevolent file which contains opcodes weight under 40000. So to keep up the proportion of malware and begin all the files fewer than 40000 loads are chosen. After this progression, 19771 and 4762 malware and benign files are left for analysis.

Preprocessing

Pre-handling Collected malware and the favorable examples can have the duplicate example (same example with the distinctive name) and that will affect the preparation. Utilizing filename to distinguish and eliminate duplicate example may prompt mistakes, consequently Message Digest (MD) 9 technique (MD5) was used to hold simply the unique example from both malware and kindhearted gathering. The proposed integrated feature set depends uniquely on PE files, for instance, EXE, DLL, etc, so it was critical to pick and pass just PE files for next stage. There are numerous strategies accessible for file type detection, among them, Linux's file utility was used to get the file kind of each example and Python content was used to channel and hold just PE files. After duplicate expulsion and PE file determination, the final dataset has 2488 benign and 2722 malware samples.

Class Labeling

The labeling phase takes as input the AV names of a, possibly huge, number of samples. For each sample, it restores a positioning of its most probable family names. This Section portrays labeling each sample. For the supervised learning, it is must to have accurate class name to every one of the example in the dataset. For naming dataset for malware detection two strategies are used recorded as a hard copy, by privately introduced AV engines or through online various AV engines. This work has chosen online multiple AV engines mode for labeling. Virus Total gives sample scanning through multiple AV engines over Internet and it likewise offers API based service which assists with robotizing the scanning process. Service to verify the class sign of each example of the dataset. Virus Total gives look at delayed consequence of different anti-virus engines running in parallel anyway this work just considered and fetches top n AV result dependent on AV-test10 result. The last decision on the class mark was made.

Feature Extraction

The objective of feature extraction is to get a set of informative and non-redundant data. It is fundamental to comprehend that features ought to represent the significant and relevant information about our dataset since without it we cannot make an accurate prediction. Toward the completion of this cycle, we expect the picked features to layout the relevant information from the underlying set so it will in general be used as opposed to starting information with no exactness misfortune. This part introduced the subtleties of the dataset utilized for both the experiments alongside the pre-processing steps and class labeling process. In the following section V. subtleties of experimental system is presented.

3.6 Random Forest Algorithm

After extracting the features, random forest algorithm is used for classification. The name on the off chance that we separate the word, it comprises of 'forest' which comprise of gathering of decision trees, and the word 'random' comes since we are doing random examining. 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 interfacing the data points to groups and sub-groups we get a structure resembling a tree, called decision tree. The algorithm at that point prepares a number of trees, resembling a forest. But each tree is unique, concerning each split in the tree, the variables are picked randomly. The leftover data set, aside 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 appeared as output. At that point, a set of labels is set to decide the kind of each application where 1denotes malware and 0 denotes favorable apps. At every node of the decision tree, it splits the training set into two subsets with various labels by minimizing the uncertainty of the class labels.

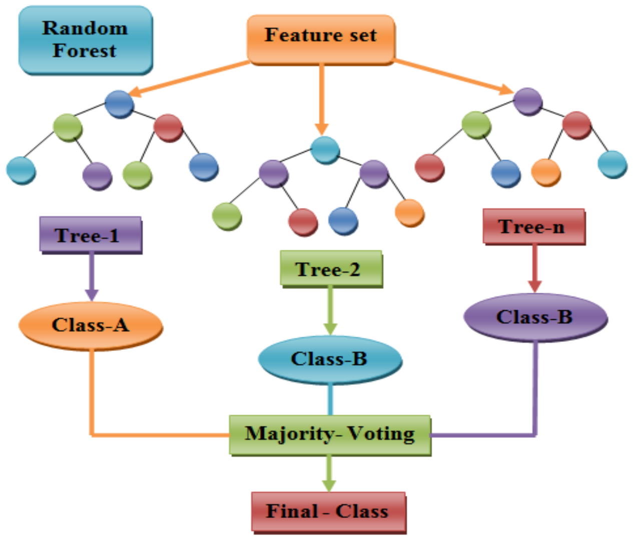


Figure 4: Random Forest Classification

3.7 Prediction

After training of dataset and discovering integrated-based features, a model is readied. A generally obscure application is sent as new data for predicting malicious of benign, the parameters of random forests at every node of every decision tree are set and have the capacity of classifying apps. At the point when any new application is gotten as input, it is decompiled using the Apktool. As examined over the source codes of the application are completely put away in smali envelope. At that point all the Raw and Derived features-related APIs, including the number of call destinations for every API, as the underlying features. The profoundly touchy APIs can be mined from a large number of integrated - related APIs via training the random forests classifier. Consequently, here these profoundly delicate APIs are used as the final features of the any new application. These features are then used as the input to acquire the result of the application type either as malware or benign.

4. EXPERIMENTAL RESULT

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

Accuracy:

Features	PIndroid	Robust malware detection system	Proposed RF-FSGA
RF	93.04	90.72	97.56
INF	90.79	92.59	98.88

Table 3: Comparison table of Accuracy

The Comparison table 3 of Accuracy Values explains the different values of existing algorithms (PIndroid, Robust malware detection system) and proposed RF-FSGA. While comparing the Existing algorithm (PIndroid, Robust malware detection system) and proposed RF-FSGA, provides the better results. The existing algorithm values start from 90.79 to 93.04, 90.72 to 92.59 and proposed RF-FSGA values starts from 97.56 to 98.88.

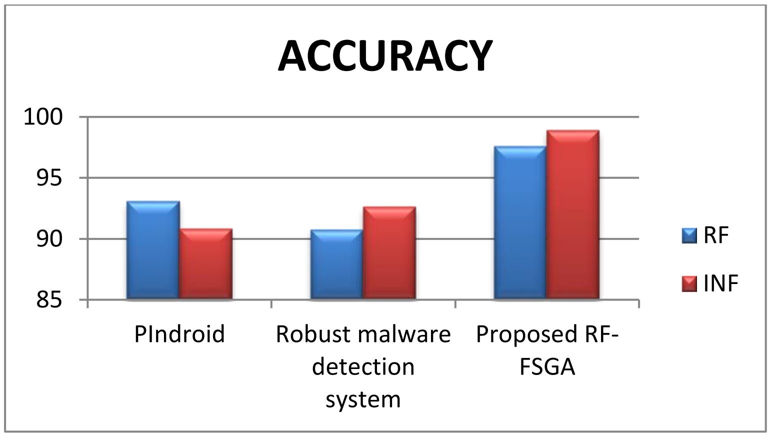


Figure 5: Comparison chart of Accuracy

The Figure 5 Shows the comparison chart of Accuracy demonstrates the existing1, existing 2 (PIndroid, Robust malware detection system) and proposed RF-FSGA. X axis denote the validation and y axis denotes the performance values in Recall. The proposed RF-FSGA values are better than the existing algorithm. The existing algorithm values start from 90.79 to 93.04, 90.72 to 92.59 and proposed RF-FSGA values starts from 97.56 to 98.88. Provides the great results.

Features	PIndroid	Robust malware detection system	Proposed RF-FSGA
RF	0.90	0.92	0.97
INF	0.91	0.93	0.99

Table 4: Comparison table of Precision

The Comparison table 4 of Precision Values explains the different values of existing algorithms (PIndroid, Robust malware detection system) and proposed RF-FSGA. While comparing the Existing algorithm (PIndroid, Robust malware detection system) and proposed RF-FSGA, provides the better results. The existing algorithm values start from 0.90 to 0.91, 0.92 to 0.93 and proposed RF-FSGA values starts from 0.97to 0.99. Provides the great results

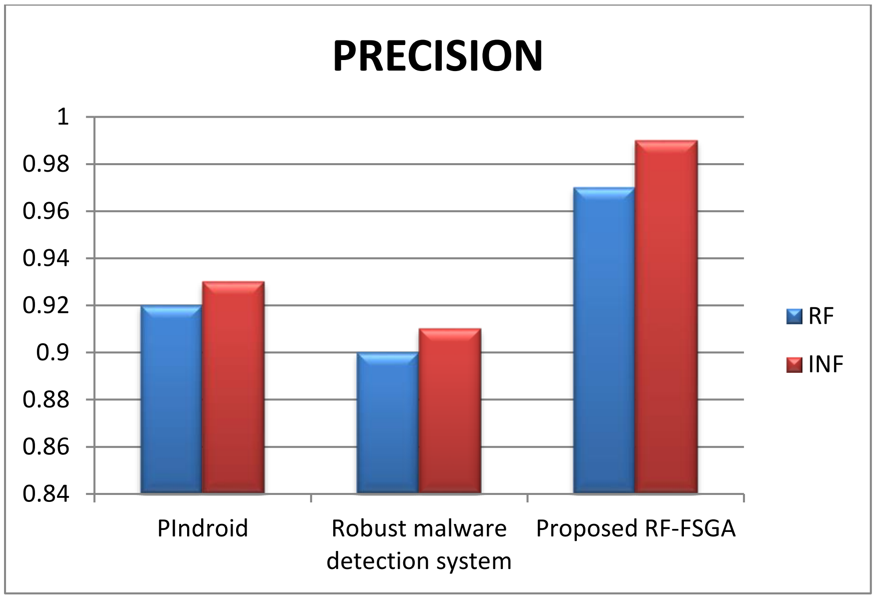


Figure 6: Comparison chart of Precision

The Figure 6 Shows the comparison chart of Precision demonstrates the existing1, existing 2 (PIndroid, Robust malware detection system) and proposed RF-FSGA. X axis denote the validation and y axis denotes the performance values in Recall. The proposed RF-FSGA values are better than the existing algorithm. The existing algorithm values start from 0.90, 0.92 and proposed RF-FSGA values starts from 0.97to 0.99. Provides the great results.

Features	PIndroid	Robust malware detection system	Proposed RF-FSGA
RF	0.90	0.92	0.97
INF	0.90	0.92	0.99

Table 5: Comparison table of Recall

The Comparison table 5 of Recall Values explains the different values of existing algorithms (PIndroid, Robust malware detection system) and proposed RF-FSGA. While comparing the Existing algorithm (PIndroid, Robust malware detection system) and proposed RF-FSGA, provides the better results. The existing algorithm values start from 0.90, 0.92 and proposed RF-FSGA values starts from 0.97to 0.99. Provides the great results

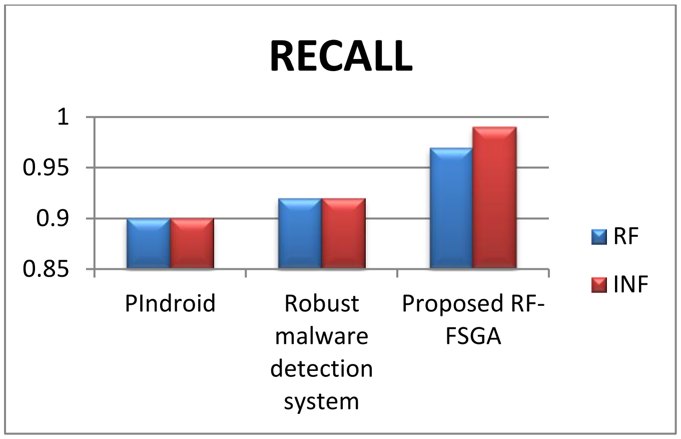


Figure 7: Comparison chart of Recall

The Figure 7 Shows the comparison chart of Recall demonstrates the existing1, existing 2 (PIndroid, Robust malware detection system) and proposed RF-FSGA. X axis denote the validation and y axis denotes the performance values in Recall. The proposed RF-FSGA values are better than the existing algorithm. 0.90, 0.92 and proposed RF-FSGA values starts from 0.97to 0.99. Provides the great results.

Features	PIndroid	Robust malware detection system	Proposed RF-FSGA
RF	0.90	0.92	0.99
INF	0.90	0.92	0.99

Table 6: Comparison table of F1- score

The Comparison table 6 of F1 – Score Values explains the different values of existing algorithms (PIndroid, Robust malware detection system) and proposed RF-FSGA. While comparing the Existing algorithm (PIndroid, Robust malware detection system) and proposed RF-FSGA, provides the better results. The existing algorithm values are 0.90, 0.92 and proposed RF-FSGA values are 0.99. Provides the great results.

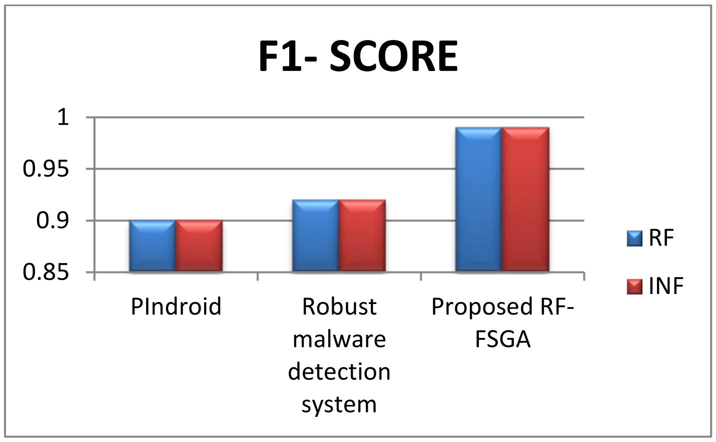


Figure 8: Comparison chart of F1 – Score

The Figure 8 Shows the comparison chart of F1 – Score demonstrates the existing1, existing 2 (PIndroid, Robust malware detection system) and proposed RF-FSGA. X axis denote the validation and y axis denotes the performance values in Recall. The proposed RF-FSGA values are better than the existing algorithm. The existing algorithm values are 0.90, 0.92 and proposed RF-FSGA values are 0.99.

CONCLUSION

Detection of malicious PE file from kind is significant as PE file format is the especially used file format, used in Windows OS. In this paper proposed work has given a novel portable executable malware detection using random forest with feature set generation algorithm (RF-FSGA) for malicious PE file detection, in like manner shows improvement in accuracy by utilizing derived features related to a subset of existing raw features over the accuracy of simply raw features. At this moment, the proposed work is restricted with PE file format anyway as the proposed feature designing technique is ordinary, it will in general be loosened up to other file formats, for instance, picture, pdf, and sound and including compact OS, for example, Android and iOS. In current work, malware and amiable samples' application type information isn't contemplated. The effect of application type on classification separation can be considered as significant future bearings, classification of PE file speaking to application type, for example, multimedia, document processing, device drivers and so forth.

REFERENCE

1. Ajit Kumara , K S Kuppusamya, , G. Aghila, “A learning model to detect maliciousness of portable executable using integrated feature set”, (2017), doi: http://dx.doi.org/10.1016/j.jksuci.2017.01.003.
2. Naser Peiravian and Xingquan Zhu, “Machine Learning for Android Malware Detection Using Permission and API Calls”, 2013 IEEE.
3. TejaswiniGhate, Chetan Pathade, Chaitanya Nirhali, Krunal Patil, Nilesh Korade, “Machine Learning Based Malware Detection: A Boosting Methodology”, ISSN: 2278-3075, Volume-9 Issue-4, February 2020.

4. Oluwakemi Christiana Abikoye, Benjamin AruwaGyunka University of Ilorin, Ilorin, Nigeria, “Android Malware Detection through Machine Learning Techniques: A Review”, https://doi.org/10.3991/ijoe.v16i02.11549.
5. Muhammad Ijaz, Muhammad Hanif Durad, Maliha Ismail, "Static and Dynamic Malware Analysis Using Machine Learning", March 2019.
6. Yaokai Feng, Hitoshi Akiyama,Liang Lu, Kouichi Sakurai "Feature Selection for Machine Learning-Based Early Detection of Distributed Cyber Attacks",Aug. 2018.
7. Qu Wei, Shi Xiao, Li Dongbao, "Malware Classification System Based on Machine Learning", September 2019.
8. Wang, T. Y., Wu, C. H., Hsieh, C. C. "Detecting unknown malicious executables using portable executable headers", IEEE, Aug. 2019.
9. Chumachenko, K. "Machine Learning Methods for Malware Detection and Classification.", 2017.
10. Kolbitsch, C., Comparetti, P. M., Kruegel, C., Kirda, E., Zhou, X. Y., Wang, X. “Effective and Efficient Malware Detection", Aug. 2009.
11. Aman, W. "A framework for analysis and comparison of dynamic malware analysis tools.", 2017.
12. Monica Kumaran, Wenjia Li, "Lightweight malware detection based on machine learning algorithms and the android manifest file", February 2018.
13. Wang Qing-Fei, Fang Xiang, "Android Malware Detection Based on Machine Learning", April 2018.
14. Anam Fatima, Ritesh Maurya, Malay Kishore Dutta, RadimBurget, Jan Masek, "Android Malware Detection Using Genetic Algorithm based Optimized Feature Selection and Machine Learning", July 2019.
15. FauziMohdDarus, Salleh Noor Azurati Ahmad, AswamiFadillahMohdAriffin, "Android Malware Detection Using Machine Learning on Image Patterns", January 2019.