Contents lists available at ScienceDirect

Applied Soft Computing Journal

journal homepage: www.elsevier.com/locate/asoc

An adaptive framework against android privilege escalation threats using deep learning and semi-supervised approaches

Shaila Sharmeen[a,1], Shamsul Huda[a,1], Jemal Abawajy[a,1], Mohammad Mehedi Hassan[b,1,*]

[a] School of Information Technology, Deakin University, Burwood, Melbourne, Australia
[b] College of Computer and Information Sciences, King Saud University, Riyadh 11543, Saudi Arabia

ARTICLE INFO

Article history:
Received 29 September 2019
Received in revised form 21 December 2019
Accepted 7 January 2020
Available online 21 January 2020

Keywords:
Mobile malware
Internet of things
Deep-learning
Semi-supervised learning
Feature extraction and selection

ABSTRACT

The immense popularity of Android makes it a primary target of malicious attackers and developers which brings a significant threat from malicious applications for android users through the escalation of the abuse of android permissions and inter-component communication (ICC) mechanism. Therefore, protecting android users from malicious developers and applications is crucial for Android market and communities. As malicious applications can hide their malicious behavior and change the behaviors frequently by abusing the android's ICC mechanism and related vulnerabilities, it is a challenging task to identify them accurately before it becomes a prevalent reason for users' privacy and data breach. Therefore, it is essential to develop such a malware detection engine that will ensure zero-day detection. In this research, we propose an adaptive framework which can learn the behavior of malware from the usage of permissions and their escalations. For our adaptive framework, we proposed two different detection models using deep learning and semi-supervised approaches. The proposed detection models can extract knowledge from unlabeled apps to identify the new malicious behavior using the unsupervised training nature of deep learning and clustering techniques and their integration to the supervised detection engine. Thus, our adaptive framework learns about new malicious apps and their behavior without supervised labeling by manual expert and can ensure zero-day protection. The proposed detection models have been tested on a real mobile malware test-bed and data set. The Experimental results show that the deep learning and semi-supervised based models achieve 99.024% of accuracies, more effective for zero-day protection and outperform other existing supervised detection engines.

1. Introduction

Android operating system has occupied the 87% market share in 2018 among all other smart phone operating systems; it becomes the most prevalent operating system for mobile and other smart devices [1–3]. More than 3 million applications have been introduced in the Android Official Apps store, Google play and nearly 65 billion downloads of these applications has been occurred [4]. In addition, unverified applications available in third-party apps stores. The flexible nature of the Android market place and the immense popularity of Android have made it a lucrative target for the malicious apps developers [5].

Android provides inter-component communication (ICC) mechanism for communication between the different apps which facilitates modular design and re-use of existing components and enables developers to make rich and user-friendly applications. However, this leaves many open doors for malicious developers to abuse android's permission mechanism. In addition, android leaves permissions setting to the users which is confusing for them due to a large number of permissions and low-level expertise. This can lead to sensitive data leakage and breach of privacy of users' data. The potential vulnerabilities of ICC and permission settings are now well known to the cyber-attackers [4] resulting an exponential growth of malicious apps in the android market place; in every 10 s, a malicious application has been released [4]. More than 9 million of malware have been found globally so far [4]. Lastly, unaware user has the control to install and use the applications from unauthorized sources. These limitations of the operating system and the smart phone devices provide the malware developers a lucrative intension to introduce new malwares which will be an alarming threat to the mobile phone industry. If the device is infected by malicious apps, then information theft can occur. The user's location, phone id, called list, contacts numbers, images, information regarding other applications, financial

* Corresponding author.
E-mail addresses: ssharmee@deakin.edu.au (S. Sharmeen), shamsul.huda@deakin.edu.au (S. Huda), jemal.abawajy@deakin.edu.au (J. Abawajy), mmhassan@ksu.edu.sa (M.M. Hassan).
1 All authors have equally participated and contributed in the paper.

https://doi.org/10.1016/j.asoc.2020.106089
1568-4946/© 2020 Elsevier B.V. All rights reserved.

credential, log in information can be breached [5]. This has a serious impact on the business organizations and the individuals. As the malware can affect a huge number of mobile devices and the users through android market place, detection of malicious apps is an urgent need to prevent the security threats for the mobile devices and their users.

Many approaches with diverse methodologies have been recognized to secure the mobile devices. Still the malware growth rate is 151% [3,4,6–10], and undeniably, we need to introduce an efficient malware detection engine to ensure zero-day protection for android platform. Due to the limited computational resources in mobile platform, general defense mechanisms of malware detection are not suitable for the mobile platforms [9]. Moreover, the mobile devices are connected to less secure networks [2] where the perimeter-based defense mechanism such as intrusion detection or firewalls are no more a reliable defense mechanism. Conventional anti-virus-based defense mechanism and host-based intrusion detection system consume more power and are not also a suitable defense mechanism for mobile platform [5]. This makes the mobile devices more vulnerable.

Although a number of static, dynamic and hybrid malware detection mechanisms have recently been introduced in the literature [3,4,6–10], still the current methods show limitations in effectiveness and comprehensiveness [2]. Static approach includes signature-based approach and permission-based analysis [11]. Most detection engines [3,4,8,10,12] applied supervised learning [11] as main classifier which can detect known malware. In the real scenario, there are a large number of new malicious apps in the market with new patterns of permission abuse. If the training database of supervised engine does not have samples of new abuse patterns, the detection engine fails to detect new malicious apps. Supervised engine requires manual effort to label the new malware by the experts which is time consuming and expensive. This leaves spaces where devices and users can be affected by new malware before the engine gets enough time to update its database.

Therefore, there is an urgent need how patterns of permission abuse can be extracted automatically from the numerous amounts of unlabeled and new malicious apps of the android market places. Towards this research question, we propose an adaptive frame work for automatic update models of detection engine. The core techniques of our framework is extraction of new attack patterns or permission abuses from unlabeled apps of the android market by using un-supervised learning. Then this knowledge will be combined with supervised detection engine to update it about the changes in new apps. We proposed two different update models using deep learning and combination of unsupervised with supervised classifier. Both deep learning and clustering can extract hidden patterns from un-supervised data which do not have any label. Deep learning techniques can learn the intrinsic hidden structure of the data through layered abstraction using multiple hidden layers through a non-linear transformation of the raw input data to a slightly higher abstraction. This capability of deep learning helps to build an adaptive detection framework for rapidly changing malicious apps in the android market place.

The main contributions of this paper are given below:

(1) A novel feature extraction and feature selection model has been developed for our adaptive framework which is later used for semi-supervised and deep learning techniques. Both the labeled and unlabeled data set are considered here to incorporate the knowledge from unlabeled data set which ensure zero-day protection.
(2) A semi-supervised method has been proposed by incorporating the unsupervised clustering to the supervised classifier which updates the database of detection engine without any manual effort.
(3) A deep learning-based detection engine has been proposed and compared with the clustering based semi-supervised approach to identify the best technique for zero-day protection. Different node arrangements in hidden layers are tested here to get the optimal performance level.
(4) A real mobile malware test-bed has been developed to extract permission abuses and test the effectiveness of our adaptive framework.

Since our model incorporates the new knowledge from the unsupervised learning, this enhances the accuracy level as well as ensure zero-day protection. Moreover, our detection model is adaptive and scalable. Manual labeling is not required here. Automatic update of malware database is also implemented in this model.

This research paper is organized as follows: The next section, Section 2 describes the efforts and achievements of the related research works. Section three presents core theories of mobile malware detection system. The proposed detection engine with deep learning-based model and semi-supervised model, data set, mechanism of feature extraction and selection have been represented in Section 4. Experimental results and comparison with other models have been depicted in Section 5. Section 6 concludes this paper by mentioning the contributions, achievements and future works.

2. Related work

As the malware detection in Android platform is a decisive need, the researchers have been carried out several works and this section will discuss these research works briefly. The literature incorporates the knowledge from these research works, identifies their major outcomes and also mentions the limitations. The static, dynamic and hybrid approaches will be represented here.

A novel approach for mobile malware classification and detection in Android systems (Zhou Q et al. 2019) [13] — A machine learning based model has been proposed to detect malware in the Android system. They have considered dynamic features and nine supervised classifying methods to compare. Only 379 malicious and 920 benign applications are considered here. They have varied the number of hidden nodes in the hidden layer and achieved 97.85% of accuracy.

Effective android malware detection with a hybrid model based on deep autoencoder and convolutional neural network (Wang W et al. 2019) [14]- 10,000 applications from Anzhi play store and 13000 malwares from Virus Share have been considered in the proposed model consists of deep autoencoder and CNN. 99.82% of accuracy has been achieved in CNN-S model whereas 98.6% of accuracy has been achieved using DAE-CNN-S model.

Deep Learning in Drebin: Android malware Image Texture Median Filter Analysis and Detection (Shi-qi L et al. 2019) [15] — Authors have been introduced an Image texture Median Filter to detect the android malware. Drebin data set is considered here. The average accuracy rate is 95.43%.

Mobile Malware Detection: A Survey (Yasmin et al. 2019) [16] Malware types and their behavior has been discussed here for mobile OS specially for Android. Signature based, Behavior based, and specification-based detection techniques have been discussed here. Comparative analysis of different classifying algorithms has been represented here.

Malware Detection Using Machine Learning Algorithms and Reverse Engineering of Android Java Code (Kedziora M et al.

2019) [17] Authors have developed a detection model using five classifiers and three selection algorithms. 1958 applications have been considered here. Among them 996 are malware. 80.7% of accuracy level have been achieved here.

MalDozer: Automatic framework for android malware detection using deep learning (Karbab EB et al. 2018) [18] — MalDozer provides minimal processing with a satisfactory level of accuracy and F1-score have been achieved using deep learning techniques and API calls are considered here as features. 90% of accuracy and 96%–99% of F1 score have been achieved here.

DroidDet: effective and robust detection of android malware using static analysis along with rotation forest model (Zhu HJ et al. 2018) [19] — Authors have proposed a malware detection model for Android platform using ensemble Rotation Forest. Here data set contains 2130 applications and the proposed method is able to obtain 88.26% of accuracy.

Deep android malware detection (McLaughlin N et al. 2017) [20] — A malware detection model has been proposed which is based on the deep CNN and static analysis. They have tested the model using small and large data set and achieved 98% and 80% of accuracy level respectively.

Droid-Sec: Deep learning in android malware detection (Yuan Z et al. 2017) [21] — Deep learning-based detection model has been proposed here. 96.5% of accuracy level is obtained with only 200 samples in test data set. Low number of test and train samples did not illustrate the real scenario of Android malware.

A multimodal deep learning method for Android malware detection using various features (Kim T et al. 2018) [22] — Authors have proposed and developed a multimodal deep learning-based model considering various static features and 41260 samples of data set with 20000 malware samples to detect the malware of Android platform. Though they have used various features and deep learning, the accuracy level is 98% of their model.

A novel android malware detection approach based on convolutional neural network (Zhang Y et al. 2018) [23] – Authors have developed a malware detection model for Android platform. They have evaluated their model using 10770 samples whereas 5546 are malware samples. Deep learning-based detection model is considered here and 97.45% of accuracy have been achieved here.

Droiddetector: android malware characterization and detection using deep learning (Yuan Z et al. 2016) [24]**-**Droid detector detects the malware using the deep learning. Here they have considered only 1760 malware samples. They have achieved 96.76% of accuracy level with considering 192 static and dynamic features.

Android Malware Detection: A Survey — A survey of Android mobile malware (Odusami et al. 2018) [5] looked at different characteristics of malware detection in Android platform. This survey describes the existing trend of malware detection process. The paper identified the primary incentives for malware detection.

- Necessity of the malware detection process
- Types of malware
- Static approaches mentioning the strength and weakness
- Dynamic approaches mentioning the strength and weakness
- Machine learning based approaches mentioning the strength and weakness
- Data set and the source
- Performance analysis using accuracy and ROC as metrics

Research Trends in Malware Detection on Android Devices — Authors (Aneja et al. 2018) [9] have discussed the comparative security analysis of iOS and Android. Among all smart phone users, 85.1% of them uses Android. The unsafe security prospects of android application distributors make the malware problem become more acute in Android platform than the iOS. They have discussed the Intrusion detection system of mobile devices and the static, dynamic and hybrid analysis using machine learning algorithms.

Fine-grained Android Malware detection based on Deep learning — Authors (Li, Wang et al. 2018) [6] have applied a deep learning-based malware detection method in the detection engine and achieved an accuracy rate of 97.16. Static features have been extracted from Manifest.xml and classes.dex file. The system has been evaluated using 5560 malware samples and 123453 benign applications.

Hybrid Android Malware Detection by Combining Supervised and Unsupervised Learning — Authors (Arora et al. 2018) [7] have proposed a hybrid malware detection system by combining supervised and unsupervised method. In the detection engine, the supervised (KNN) method and the unsupervised (K- medoids) method is combined and provide an accuracy rate of 91.98%. However, an improvement in accuracy level is still required. Moreover, the numbers of samples in both training and testing data set are relatively low.

FgDetector: Fine-grained Android Malware Detection — Authors (Li, Wang et al. 2017) [3] have presented a fine-grained malware detection mechanism for Android. It has used the Drebin data set with 5560 malwares from different families. Various classifying algorithms such as Gaussian Bayes, Bernoulli Bayes, Decision Tree, Random forest, Logistic Regression and SVM have been considered in the detection engine. Random forest has shown highest accuracy which is 98.61%.

Significant Permission Identification for Machine-Learning-Based Android Malware Detection — Authors (Li, Sun et al. 2018) [4] have introduced three levels pruning techniques named permission ranking with negative rate, support based ranking and permission mining with association rules to select the most suspicious permissions. They have mentioned top twenty-two permissions which are used to train and test the data sets consists of 2650, 5494 and 54694 malwares. An accuracy level of 93.62% has been achieved.

Android Malware Detection Using Parallel Machine Learning Classifiers- Authors (Yerima et al. 2014) [8] have introduced a parallel machine learning based mechanism where they have used five different classifying algorithms as a composite model of heterogeneous classifiers. They have combined the outcomes of these classifiers by measuring the average of probabilities, product of probabilities, maximum of probabilities and majority vote. Among these schemes, the maximum of probabilities scheme provides the best accuracy level. They have achieved 98.6% of accuracy by using the aforementioned schemes.

High Accuracy Android Malware Detection Using Ensemble Learning- Authors (Yerima et al. 2015) [10] have introduced an approach which involves static analysis and machine learning techniques. Enhanced performance have been achieved by considering ensemble learning. Detection system has considered different classifying algorithms such as Simple Logistic, Decision tree, Random tree, Random Forest etc. The highest level of accuracy is 99.3% which is achieved by the random forest using 179 features. They run this experiment on 2925 malware and 3938 benign wares.

RiskRanker: Scalable and Accurate Zero-day Android Malware Detection-(Grace et al. 2012) [12] A proactive malware detection mechanism, Risk Ranker has been proposed to ensure Zero-day detection. Risk ranker has developed a ranking scheme to categorize the applications into high, medium and low risk category. The have examined 118,318 apps and among them they found total 281 risky applications. 718 malwares have been detected and among them 322 are discovered as zero-day detection.

MADAM: a Multi-Level Anomaly Detector for Android Malware — (Dini et al. 2012) [25] MADAM is a multi-level detection model where features are extracted from kernel level, application level and package level. Authors have considered static as well as dynamic features. MADAM has divided the detection process into three main activity blocks-App Risk Assessment, Global Monitor and per-app Monitor. 96.9% of accuracy has been achieved in detecting the malware. The main strength of MADAM is to use multi-level of features.

DroidMat: Android Malware Detection through Manifest and API call Tracing — Authors (Wu et al. 2012) [26] have proposed a malware detection technique depending on static features. API calls, intents and components are collected from manifest and smali files. k-means algorithm is used as the classifying algorithm and singular Value Decomposition (SVD) has be applied to decide the cluster numbers. DroidMat has achieved 97.87% of accuracy.

Android: From reversing to Decompilation – Authors (Desnos et al. 2011) [27] have introduced a new algorithm to calculate the similarity and dissimilarity between two applications. They have implemented a new open-source tool (Androguard) written in Python to do the reverse engineering of the Android applications. The accuracy level of 93.04% has been achieved here.

Using Probabilistic Generative Models for Ranking Risks of Android Apps — Authors (Peng et al. 2012) [28] have introduced a model to analyze the risk communication using risk scoring and risk ranking. Probabilistic generative models for risk ranking have been introduced. They have collected a huge number of benign applications where as only 378 malwares are considered. Achieved accuracy level is 88.2% and the false positive rate is 1.71%.

The analysis of feature selection methods and Classification algorithms in Permission based Android Malware Detection — Authors (Pehlivan et al. 2014) [29] have examined different feature selection methods: Gain Ratio Attribute Evaluator, ReliefF Attribute Evaluator, Cfs subset Evaluator and Consistency Subset Evaluator. Random forest classifying algorithm [30] shows the highest accuracy of 94.90% using 25 selected permissions using Cfs Subset selection method.

Derbin: Effective and explained detection of Android malware in your pocket — Authors [31] have represented a lightweight detection engine. A large number of features such as hardware components, requested permissions, App components, filtered intents, used permissions, Restricted API calls, suspicious API calls and Network addresses have been considered. They have achieved 93.8% of accuracy level using SVM.

A two layered Permission based Android Malware detection scheme- Authors [32] have been introduced a two-layered based detection engine which identify the malware. Requested and required permissions are considered here as features. The number of benign applications is 28548 and 1536 malware are used here. They have considered j48 as their detection algorithm and achieved 98.6% of accuracy.

Machine learning based hybrid behavior models for Android malware analysis — Authors [33] have designed a hybrid model based on the static features. Features (malicious preferred and normal-preferred) and SVM as the classifying algorithm have been considered here. It is based on static analysis where permission and API call are extracted as features. 6005 Benign applications and 3368 Malicious applications are tested here, and accuracy level is 96.69%.

ICCDetector: ICC-Based Malware detection on Android- Authors [34] have considered the Inter Component Communication features to detect the malware whereas the conventional detection methods considered the required resources as the features such as permission, API call, system call etc. They have evaluated 5264 malwares and 12026 Benign wares using their model ICCDetector and achieved 97.4% of accuracy level.

3. Core theories of mobile malware detection system

The malware detection system is a complete mechanism which will identify the malware and Fig. 1 shows the functional modules of the detection engine. The detection engine consists of a set of malware and benign ware (described in Section 4.3), feature extraction module (described in Section 4.4), feature vector generation module (Algorithm 3), feature selection module (described in Section 4.4) and the detection module (described in Section 4.5). Android allows third party market places. Applications are downloaded from these marketplaces and the official market place, Google play. In the detection engine, the data set module refers these malwares and the benign wares. The feature extraction module extracts the required features which will represents the characteristics/behavior of the application. Based on the nature of the feature extraction, features are named as static, dynamic and hybrid features. Execution of the application is needed in dynamic feature collection process where as it is not required in static feature collection process. Hybrid features are the combination of static and dynamic features. Feature extraction module extracts a huge number of features. Among them, some features play significant role in the process of malware detection where as some are not relevant to the detection process. The feature selector module selects features which are more relevant and significant than other features. If the relevant features are selected, then this will provide a positive impact on the achieved accuracy level of the detection engine and minimize the computational cost of the malware detection process. Feature Vector is generated using the features in the row and applications in the column. Feature vector is used in desired format in the classifying module. The classifier will classify the applications into two groups: malware and benign ware. Different classifying algorithms can be used in the classifying module such as Naïve based, random forest, j48, random tree, SVM etc. Fig. 1 represents the malware detection system.

3.1. Malware detection problem

Among the applications, there are two types of applications-malware and benign ware. Our research goal is to identify the malware. We need to classify these applications into two groups depending on the features or characteristics. Since Binary classification classifies the objects into two classes, our malware detection process suits this Binary classification. We can describe our malware detection process as a binary classification problem. Suppose that, various applications form a set of application, $A = \{A_1, A_2, A_3, \ldots, A_n\}$ where there are n number of applications. Features are extracted and selected from these applications and the feature vector $F_i = \{F_1, F_2, F_3, \ldots, F_k\}$ is generated with k numbers of selected features. We can express each application in terms of feature vector and class level CL_i using the following equation:

$$A_i = \{F_i, CL_i\} \tag{1}$$

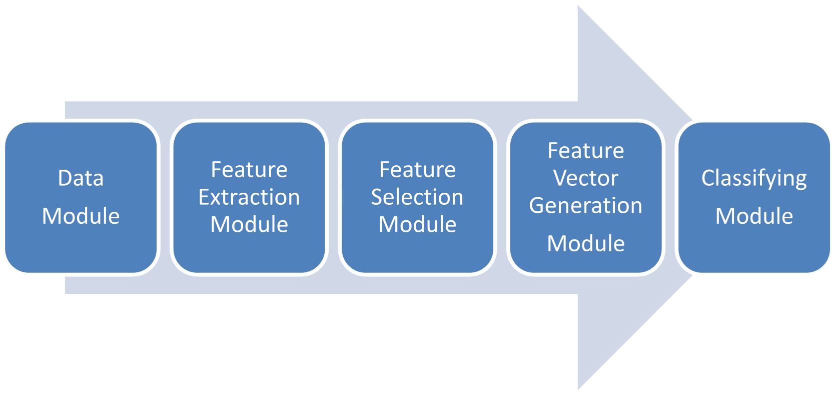


Fig. 1. Malware detection system.

Class label CL_i represents the class of the applications. Since we have considered the unlabeled data as well as the labeled data, we notify the applications using three labels: benign ware CL_b, malware CL_m and unknown applications CL_{uap}. The job of the detection engine to label the unknown applications and identify the malware if there is any. The detection engine will label the applications depending on the features. The detection engine will consider classifying algorithm to classify the set of apps A and label them as CL_b or CL_m. The Malware detection function can be expressed as follows:

$$\mu(F): F \rightarrow CL \tag{2}$$

3.2. Supervised learning approach in detection system

Machine learning involves a learning process that will set rules to take a decision from provided instances and this knowledge will be applied to the generalize/categorize a new instance. The supervised learning deals with the labeled data. The first step of supervised learning is data collection, we have to collect the labeled data for the supervised learning. Brute-force data collecting method is usually used [11]. As the data contains noise and missing values, the pre-processing is needed. Collected data has been processed to get the desired and most informative information which are described by the term features. Then Feature Selection method is applied to eliminate the unnecessary and comparatively less relevant features. Feature Selection reduce the amount and dimensionality of the data by removing irrelevant features and enable data mining approaches to find out the optimal solutions with more accuracy [11]. After the feature selection, we get a feature vector. As supervised learning deals with labeled data, the classifier is trained based on feature vector of the training data set. Then the test data set is applied, and trained model is examined and tested. The malware detection system finds out the class of the test data set. Since labeled data is considered in supervised learning, we compare the new classification result and the previous labeling of data. Thus, we calculate the accuracy of the detection engine. However, the concern is that the labeled data is rare in our real life and labeling of the data is a costly and time-consuming process. Lacking labeled data is the main limitation of supervised learning. Supervised learning method is illustrated in Fig. 2.

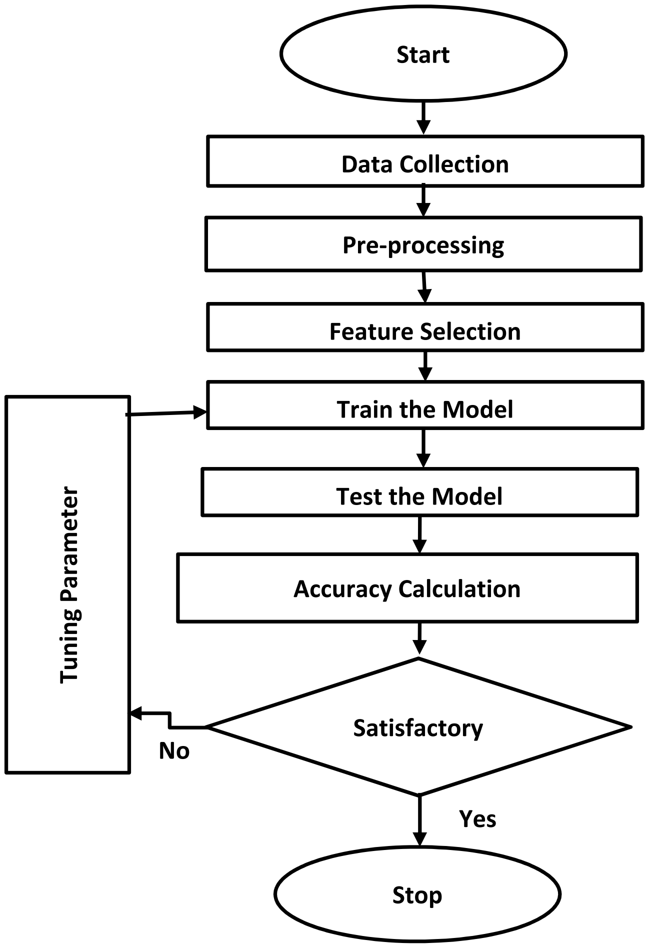


Fig. 2. Learning process in supervised detection method.

3.3. Unsupervised learning for detection system

The unsupervised learning deals with the unlabeled data while supervised learning deals with labeled data. Unlabeled data is easily available and accessible. In real life context, a huge volume of unlabeled data can be managed within a short time period where as the getting the labeled data needs more processing, resources and time. Unsupervised learning is traditionally known as Clustering. Given data set are categorized in different clusters depending on the similarities of their properties such as features. In the very first step, we need to collect the data. The clustering algorithm will find the cluster of the object depending on the relevant features. This is why, we need to do the feature extraction and feature selection to collect and select the relevant features. In real life, the history or the background knowledge can be applied to select the set of features [35]. Clustering can be called unsupervised as it considers the unlabeled data and these data are clustered into a finite set of natural groups. When the clustering is done, we have to consider the high similarities among the group members and high dissimilarities among the members of other groups. K-means clustering algorithm divides the data set into k number of groups depending on the similarities of the characteristic of the object and widely used since 1967. The similarities can be calculated from different mathematical and proximity measures. K-means algorithm works as follows [11,35, 36]:

Algorithm 1: k-means Clustering

```
1. input ← UDL($F_1F_2, F_3 \ldots\ldots\ldots\ldots F_m$) {Unlabeled data set with m number of global features}
2. Output Clusters
3. Begin
4. Initialize the cluster centers by choosing them randomly
5. The distance from each data point to chosen cluster centers has been measured using different difference calculation equations stated below. Here $X_i$ represents the data points.
```

$$Euclidean\ Distance D_{xy} = \sqrt{\sum_{k=1}^{d} \left(X_{ik} - X_{jk}\right)^2} \tag{3}$$

$$Manhattan\ Distance D_{xy} = \left|X_{ik} - X_{jk}\right| \tag{4}$$

$$Minkowski\ Distance D_{xy} = \left(\sum_{k=1}^{d} \left|X_{ik} - X_{jk}\right|^{\frac{1}{p}}\right)^{P} \tag{5}$$

```
6. The minimal distance from cluster centers will determine the cluster of that data point.
7. Update the cluster centers using following equation where $C_i$represents the number of data points in the cluster, $X_i$ represents the data points, $V_i$ represents the cluster centers.
```

$$V_i = \left(\frac{1}{C_i}\right) \sum_{1}^{cl} x_i \tag{6}$$

```
8. The distance from each data point to updated cluster centers have been measured.
9. Repeat the steps from 6 to 8 until all the data points are assigned to a cluster and no updates observed.
```

The above-mentioned clustering algorithm shows Euclidean distance, Manhattan distance and Minkowski distance as the distance measuring procedures. The distance is measured from the instance to the cluster centers. Since we are going to apply this algorithm in malware detection, these distance measuring approaches will not be considered here. As we considered the API call, permissions, API call frequencies as the features, we need to adapt different approaches for measuring the distance. Term Frequency * Inverse Document frequency can be used to measure the importance of the term in the document [37]. TF*IDF will express the relevancy of a term in a document [37–39]. In case of malware detection, we need to identify whether or not the permissions, API calls are relevant to the detection of the malware.

Term-Frequency is the frequency of a word in the document and IDF will measure the weight of the rare words. High IDF score will represent that the word is rare in the document. Let, we have N number of documents and a word is occurred in n times in these N documents.

$$Term\ Frequency, tf_i = \frac{n_i}{\sum_k n_i} \text{ where } n_i \text{ is the occurrence of a word in a document} \tag{7}$$

$$Inverse\ Document\ Frequency, \text{IDF} = \log\frac{N}{n} \tag{8}$$

$$\text{TF–IDF } score, w_i = tf_i \times \text{IDF} = tf_i \times \log\frac{N}{n} \tag{9}$$

If we consider the malware detection scenario then we can consider the followings and find out the TF–IDF score as follows,

$N = number\ of\ applications$

$n_t = number\ of\ occuranceof\ tth\ API$

TF–IDF score of the *ith* application can be written as follows,

$$\text{TF–IDF}_i = \frac{1}{\sqrt{tf_1^2 + tf_2^2 \cdots + tf_t^2}} \times [tf_1 \log\frac{N}{n_1} tf_2 \log\frac{N}{n_2} \cdots . tf_t \log\frac{N}{n_t}] \tag{10}$$

$$\text{TF–IDF}_i = [w_i^1 \times tf_1^i w_i^2 \times tf_2^i \cdots . w_i^t \times tf_t^i] \quad \text{where} \quad w_i^t = \frac{\log\frac{N}{n_t}}{\sqrt{tf_1^2 + tf_2^2 \cdots + tf_t^2}} \tag{11}$$

Cosine similarity is the distance measuring mechanism while we are working on text data and the occurrence of a word is accounted [40]. If there are two documents with $\overrightarrow{r_a}$ and $\overrightarrow{r_b}$ m-dimensional vectors over term set $R = \{r_1, r_2, \ldots . r_m\}$ then the cosine similarity can be measured as follows,

$$cosinesimilarity\left(\overrightarrow{r_a}, \overrightarrow{r_b}\right) = \frac{\overrightarrow{r_a} . \overrightarrow{r_b}}{\left|\overrightarrow{r_a}\right| \times \left|\overrightarrow{r_b}\right|} \tag{12}$$

We consider the applications $a1$ and $a2$, we will find out the TF–IDF score for these two applications and these will help to find out the cosine similarities of these applications.

$$\text{TF–IDF}_{a1} = [w_1^1 \times tf_1^1 w_1^2 \times tf_2^1 \cdots . w_1^t \times tf_t^1]$$
$$\text{TF–IDF}_{a2} = [w_2^1 \times tf_1^2 w_2^2 \times tf_2^2 \cdots . w_2^t \times tf_t^2]$$
$$cosine\ similarity\left(\overrightarrow{a1}, \overrightarrow{a2}\right) = \frac{\overrightarrow{\text{TF–IDF}_{a1}} . \overrightarrow{\text{TF–IDF}_{a2}}}{\left|\overrightarrow{\text{TF–IDF}_{a1}}\right| \times \left|\overrightarrow{\text{TF–IDF}_{a2}}\right|} \tag{13}$$

To incorporate the k-means clustering algorithm in our problem domain, cosine similarity will be applied as the distance.

3.4. Deep learning based approach in detection system

Deep learning-based detection model can be applied in the Android mobile malware detection system [15,18,20–24]. Generally, in a deep learning-based system, there are input layer, hidden layer and output layer [15]. Different hidden layers with various number of nodes can generate different level of abstraction of the input data and that can be applied to the computational model. The sophisticated structure of the data set can be structured and recognized by the deep learning-based method [24]. This nature of deep learning can be applied in the malware detection process to identify the malware. In the detection process, we have the data set with features and need to find out the classification of that data set. These features will generate the feature vector. In deep learning-based method, these feature vector will be played the role of input layer. The hidden layers will represent the higher level of abstraction of data depending on the information of the lower layer [23]. Thus, a non-linear transformation of data will be occurred in the hidden layer. Depending on the knowledge from the hidden layers, the output layer will provide the output. Since we will apply the deep learning in classifying the malware, the output layer will provide whether it is a malware or not. The loss function will be calculated to determine the performance of the learning process [15].

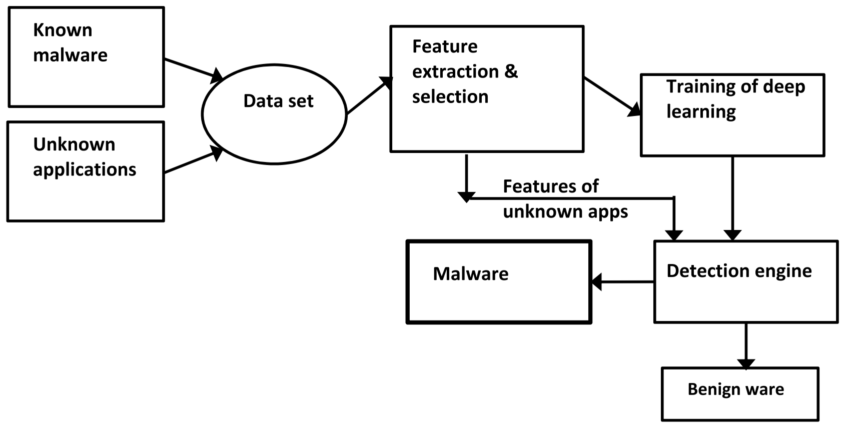


Fig. 3(a). Android malware detection engine using deep learning.

4. Proposed approaches for malware detection system

Due to sophisticated obfuscation methods, detection of malware remains a challenging endeavor. The Malware developers have contributed a lot in developing and hiding the malicious activities using sophisticated and diverse techniques. The current malware detection processes are facing difficulties in detection of the malware as the real characteristics of malware/benign ware cannot be obtained from only using limited labeled data. We propose an adaptive framework to capture the dynamic nature of attack patterns from the wild malicious apps which are coming to the android market place every day and are unlabeled. Two different types of detection models have also been proposed. These are based on unsupervised learning and their combination with supervised learning. Proposed approaches have been described in the next two sub-sections.

4.1. Proposed detection model-1: Using supervised deep learning approach

In first proposed approach, we propose a deep learning based the detection process. Deep Learning can be trained by extracting the features and patterns of the malware through multiple hidden layers. As the malware developers have introduced many modern techniques to hide the malicious and suspicious activities, supervised deep learning can be applied in this detection process. The hidden layers and the nodes in each layer can represent the actual attack patterns to a non-linear and higher abstraction of the real scenarios and make the detection process more accurate. Fig. 3(a) has illustrated the proposed model. We have constructed our data set by collecting labeled and unlabeled data set. The labeled data set contains known malware and known benign ware. The applications that are downloaded from different online sources are considered as the unlabeled data set. After collecting the data set, we have extracted the features and the relevant features are selected. The model is trained using the features from labeled data set and deep learning-based method is applied here. The detection engine will be tested using the test data set. If the detection engine will find any new malware, we will store it in the database.

In our detection model, we have considered two hidden layers. Since three or more hidden layers need more computational time, we considered two hidden layers. We have adapted various architecture of nodes by varying the number of nodes in each layer [41]. We have proposed three different architectures of node arrangement in our model. The first one is the Deep learning-based model with equal number of nodes in layer 1 and layer 2, the second one is a smaller number of nodes in layer 1 and more nodes in layer 2 and the third one is the more nodes in layer 1 and less nodes in layer 2. The following Table 1 shows the detailed architecture of the node arrangements.

Each RBM layer represents two layers in the deep learning. If we denoted the layers as "i" and the nodes in the hidden layer as "j" then the structure of input (v) and hidden nodes (h) holds the energy [42] as follows

$$E\left(v,h\right)=-\sum_{i}a_iv_i-\sum_{j}b_jh_j-\sum_{ij}v_ih_jw_{ij} \tag{14}$$

Where v represents the input (visible nodes), h represents hidden nodes, a is the bias of input nodes, b is the bias of hidden nodes and w is the weight.

Each node will generate 0 or 1 according to the following equation

$$P\left(v_{i=1}|h\right)=\sigma(a_i+\sum_{j}w_{ij}h_j) \tag{15}$$

$$P\left(h_{j=1}|b\right)=\sigma(b_i+\sum_{i}w_{ij}v_i) \tag{16}$$

In these equations, $\sigma\left(x\right)=\frac{1}{(1+e^{-x})}$ where $x=a_i+\sum_j w_{ij}h_j$ and $x=b_i+\sum_i w_{ij}v_i$ respectively. A stochastic ascent algorithm is followed (Eq. (17)) in tanning section and the weight update rule is managed (Eq. (18)) according to the followings:

$$p\left(v\right)=\frac{1}{x}\sum_{j}e^{-E(v,h)} \tag{17}$$

$$\text{Change of weight matrix, } \Delta w_{ij}$$
$$=\varepsilon(\langle v_ih_j\rangle data-\langle v_ih_j\rangle reconstruction) \tag{18}$$

Here, $\langle v_ih_j\rangle data$ is the expectation of data and $\langle v_ih_j\rangle reconstruction$ is the expectation of reconstruction distribution and learning rate ε should be more than zero.

Moreover, we have varied the number of epochs. We have varied the epoch number from 50 to 500. We have considered the ten-fold cross validation and the structure with best performance is considered for our detection model. The architectures of nodes that are considered for our model is given below:

4.2. Proposed detection model-2 using semi-supervised approach

The semi-supervised method integrates the benefits of supervised and unsupervised learning process. As a result, the semi supervised approach provides some specific advantages. Semi supervised approach facilities the detection process by using both labeled and unlabeled data. It makes the detection process more

Table 1
Different Structure of nodes.

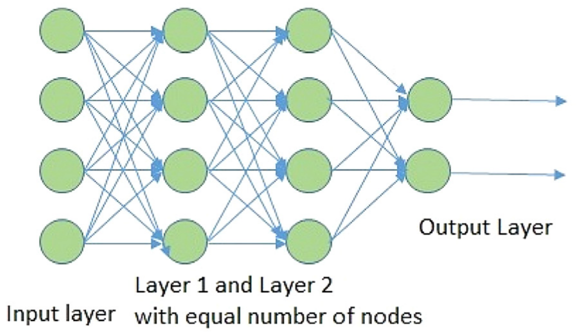
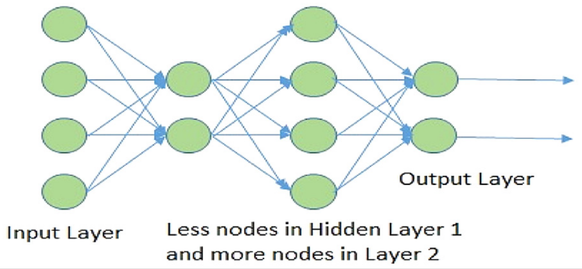
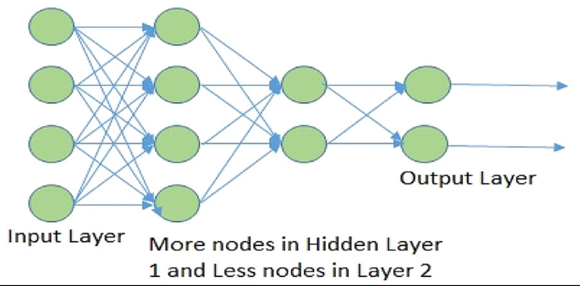
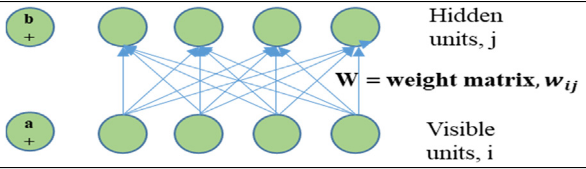
Type	Diagram
Equal number of nodes in layer 1 and layer 2	 Output Layer Layer 1 and Layer 2 with equal number of nodes Input layer
Less number of nodes in layer 1 and more nodes in layer 2	 Output Layer Input Layer Less nodes in Hidden Layer 1 and more nodes in Layer 2
More nodes in layer 1 and less nodes in layer 2	 Output Layer Input Layer More nodes in Hidden Layer 1 and Less nodes in Layer 2
Restricted Boltzmann Machine (RBM)	 b + a + Hidden units, j W = weight matrix, w_{ij} Visible units, i

Table 2
Architecture of nodes.

Type	Definition	Notation
Equal number of nodes in layer 1 and layer 2	16 nodes in each layer	L16_L16
	32 nodes in each layer	L32_L32
	64 nodes in each layer	L64_L64
	128 nodes in each layer	L128_L128
	512 nodes in each layer	L512_L512
	1024 nodes in each layer	L1024_L1024
Less number of nodes in layer 1 and more nodes in layer 2	16 nodes in layer 1 and 32 nodes in layer 2	L16_L32
	16 nodes in layer 1 and 64 nodes in layer 2	L16_L64
	32 nodes in layer 1 and 64 nodes in layer 2	L32_L64
	512 nodes in layer 1 and 1024 nodes in layer 2	L512_L1024
More nodes in layer 1 and less nodes in layer 2	32 nodes in layer 1 and 16 nodes in layer 2	L32_L16
	64 nodes in layer 1 and 16 nodes in layer 2	L64_L16
	64 nodes in layer 1 and 32 nodes in layer 2	L64_L32
	1024 nodes in layer 1 and 512 nodes in layer 2	L1024_L1024

accurate, effective and faster than the supervised method. Usually researchers choose supervised process in malware detection process. The detection model is trained and tested using the labeled data set. At the very beginning, the features are extracted from the labeled data. Then selected features are then used in the detection process. The accuracy of the detection is measured using the comparison of actual labeling of test data and the determined labeling by the classifier.

The supervised method suffers from specific limitations as the labeling of the data is quite expensive and time consuming [11]. The labeled data set is limited in amount and hard to collect. Sometimes labeling becomes impossible due to cost and limited available resources. Moreover, we cannot assure the

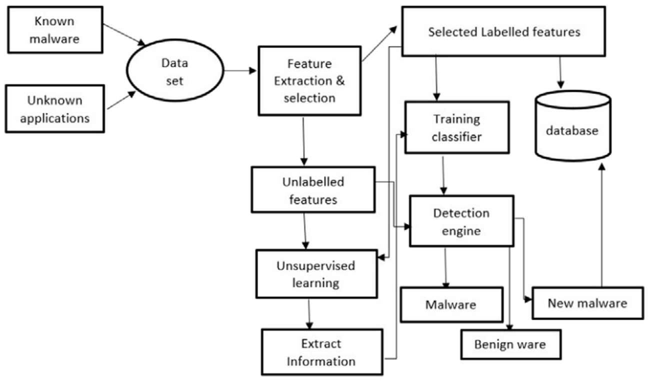


Fig. 3(b). Android Malware detection engine-using semis-supervised model.

performance of our detection model as the test data is labeled and do not depict the real data. The unsupervised learning does the classification using the process of clustering in such a way that the similarities within the cluster is maximized whereas the similarities among groups is minimized. The unlabeled data can be used in unsupervised learning. Availability of the unlabeled data is a major advantage of unsupervised clustering. Unsupervised classification put the same type of data in a group and the patterns within the group have high similarities [43]. The similarities within the group member is high and the dissimilarity with the members of another group is also high. Minkowski distance [35,36], Maximum Likelihood [43], expectation maximization etc. are used to extract the intrinsic patterns. As we are dealing with malware classification, a binary classification problem, so the unsupervised classification will not be appropriate here. However, the intrinsic behavior from unsupervised learning can play a vital role in supervised malware detection process. We can define the number of clusters as binary and measure the distance of each instance from the cluster centers. This measured distance will show the amount of similarities of characteristics of an application to cluster center. If it is more similar to the malware cluster, then it can be a malware, otherwise it will be a benign ware. In our proposed algorithm, we have integrated this measured distance to the feature vector. In our detection model, first we train the classifier using the labeled data, then the unlabeled data is provided to the classifier as test data set. The classifier generates a pseudo labeling of the unlabeled data. Then the classifier is again trained with the new train data set including the pseudo labeled data. Thus, the classifier enhances the performance of the detection engine. Moreover, as we have considered unlabeled data along with labeled data, it will represent the data set which is alike the real-life applications and can handle the new unknown behavior of latest malware.

The reasons why we have considered the unsupervised and supervised approaches and proposed the semi-supervised model will be presented here. Firstly, we can illustrate the real nature of the new malwares only if we considered the unlabeled data and unsupervised clustering. The knowledge from the unsupervised clustering then inserted into the feature vector. Thus, the intrinsic patterns of the applications can be obtained and helpful to recognize the malware. Secondly, the labeling cost can be avoided. Thirdly, the incorporation of knowledge of unsupervised method to supervised one will be the key strength of the model in detecting the new malware. Since the behavior of the malware changes very frequently, this semi-supervised model will be able to detect new malwares by incorporating the knowledge from unsupervised clustering.

Table 3
Top twenty malware family details [31].

FakeInstaller	Adrd
DroidKungFu	DroidDream
Plankton	LinuxLotoor
Opfake	GoldDream
GingerMaster	MobileTx
BaseBridge	FakeRun
Iconosys	SendPay
Kmin	Gappusin
FakeDoc	Imlog
Geinimi	SMSreg

In our proposed model 2 (Fig. 3(b)), the semi supervised approach has been considered and here we have united the supervised and unsupervised machine learning in order to accumulate the strength of these methods to produce a single classifying mechanism. The second proposed detection model has been presented in Fig. 3(b). The semi-supervised model is trained using the features from labeled data set. We remove the levels of the labeled data and we have constructed a new data set by combining the labeled (after removing labels) and unlabeled data. Then we applied unsupervised clustering on this new data set. The unsupervised clustering finds out the two cluster centers and measure the distance from two cluster centers. These distances represent the intrinsic behavior of each application. We have injected these distances in the feature vector. Thus, we append the strength of unsupervised learning into the supervised learning. After appending this knowledge, we train the model again with the help of new feature vector. Then eventually, we test the model and calculate the accuracy of proposed model. As the training process has incorporate the knowledge from unsupervised clustering, the accuracy of the detection engine will be enhanced and the probability of finding new malware will be increased.

4.3. Data collection

Since our experiments need real android malware and benign ware, we collected the malware and benign ware from online resources from the years 2010–2018. Data is collected from four sources: Drebin data set, Androzoo, ApkPure and ApkMirror and the collection period is April, 2018. Total Labeled data is 18560 and unlabeled data is 11956. There are 8560 malware samples from 179 different malware families. The malware families have been represented in Table 3. Collected 5560 malwares from Drebin data set [31] and they are collected during the year 2010–2012. To integrate new and modern malwares till 2018, we have collected more malware from Androzoo [44]. 10000 benign applications are collected from Drebin data set [31]. More Android applications are collected from ApkPure [45] and ApkMirror [46]. A crawler application has been written to download these applications from ApkPure and ApkMirror. We have considered other sources. However, crawler application cannot run in most of the cases. 1956 applications from the ApkPure [45] and ApkMirror [46] will be served as the unlabeled data. 10000 unlabeled applications are collected from Androzoo [44]. Table 4 will describe summary of the source, number and nature of the applications.

4.4. Feature extraction and selection

A python script has been written using the functions of Androguard [47] to extract features such as API call and the permissions and generates a global set of features. In order to minimize the computational cost, we have considered the static features only.

Algorithm 2: Feature Extraction and Global Feature Set Generation

```
1.  $input \leftarrow Labeled\ (Malware\ and\ benignware) and\ Unlabeled\ Apk\ dataset$
2.  Output $Global\ Feature\ Set$
3.  Begin
4.  for each apk in input data set do
5.      Decompile the apk
6.      Read manifest.xml to get the permissions and store in database
7.      Parse all java files and use regular expression to find API call and store in database
8.  end for
9.  $Global\ Feature\ Set \leftarrow \emptyset$
10. for each $perm$ in all stored permissions in database do
11.     if $perm$ not in $Global\ Feature\ Set$ then
12.         $Global\ Feature\ Set \leftarrow perm$
13. end for
14. for each $API$ in all stored API calls in database do
15.     if $API$ not in $Global\ Feature\ Set$ then
16.         $Global\ Feature\ Set \leftarrow API$
17. end for
18. return $Global\ Feature\ Set$
19. END
```

Algorithm 3: Feature Vector Generation

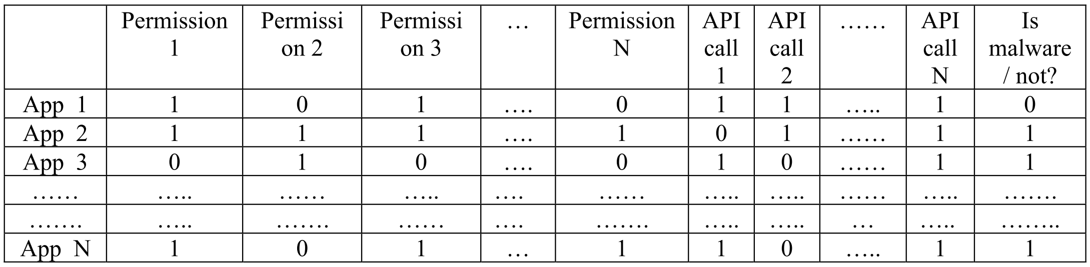
```
1.  $input \leftarrow Labeled\ (Malware\ and\ benignware) and\ Unlabeled\ Apk\ dataset$
2.  $input \leftarrow Global\ Feature\ Set, Feature\ Groups$
3.  Output $Feature\ vector\ matrix$
4.  Begin
5.  $Feature\ vector\ matrix \leftarrow \emptyset$
6.  for each apk in input data set do
7.      $permissions \leftarrow all\ permissions\ from\ database\ for\ this\ apk$
8.      $API\_Call \leftarrow all\ API\ Calls\ from\ databas\ for\ this\ apk$
9.      $APK\_Features \leftarrow concat(permissions, API\_Call)$
10.     $Row \leftarrow \emptyset$
11.     $Row['is\_malware'] \leftarrow$
            $\{0\ for\ known\ benign\ apk, 1\ for\ known\ malware\ apk, '?'\ for\ unlabelled\ apk\}$
12.     for each $Feature$ in $Global\ Feature\ Set$
13.         if $Feature$ in $APK\_Features$ then
14.             $Row[Feature] \leftarrow 1$
15.             for each $Group$ in $Feature\ Groups$
16.                 if $Feature$ in $Group$ then
17.                     $Row[Group] \leftarrow Row[Group] + 1$
18.             end for
19.         else
20.             $Row[Feature] \leftarrow 0$
21.     end for
22.     Append $Row$ to $Feature\ vector\ matrix$
23. end for
24. return $Feature\ vector\ matrix$
25. END
```

Table 4
Data set details.

Nature	Number	Source	Total
Malware	5560 3000	Drebin data set [31] Androzoo [44]	8560
Benign	10000	Drebin data set [31]	10000
Labeled applications		Malware and benign ware	18560
Unlabeled applications	10000 1956	Androzoo [44] ApkPure and ApkMirror [45,46]	11956

The feature extraction process and global feature set generation process are represented in the following Algorithm 2. To generate the feature vector, we use the binary denoting for the existence of permissions and API call. If a permission is denoted by P, then the application APK is represented by $x = (0, 1, 1, 0 \ldots 1, 0 \ldots) \in \{0, 1\}^p$ where 1 represents the existence of the permission and 0 represents the absence of the permission. Thus, we have generated a two-dimensional matrix (Fig. 4), where the row denotes the application and column denotes the API call and the permissions. The detailed feature vector generation process is described in Algorithm 3.

Filter and wrapper are two common methods in feature selection [48–50]. In our model, we have considered the wrapper



	Permission 1	Permissi on 2	Permissi on 3	…	Permission N	API call 1	API call 2	……	API call N	Is malware / not?
App 1	1	0	1	….	0	1	1	…..	1	0
App 2	1	1	1	….	1	0	1	……	1	1
App 3	0	1	0	….	0	1	0	……	1	1
……	…..	……	…..	….	……	…..	…..	……	…..	…….
…….	…..	…….	……	….	…….	…..	…..	…	…..	……..
App N	1	0	1	…	1	1	0	…..	1	1

Fig. 4. Feature vector.

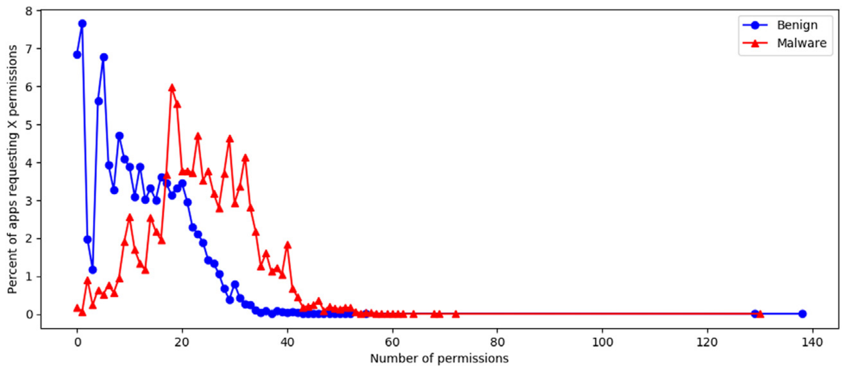


Fig. 5. Percentage of application requesting permissions (using global set of permissions).

method. The Boruta algorithm has been considered as the wrapper method to select the best suitable features [48]. Boruta [48] is a feature selection method that will find all relevant variables using wrapper filter model for Random Forest classification algorithm. This is a statistical method that will remove the less relevant features by using statistical tests. By using this wrapper-based feature selection model, we have selected most relevant permissions and API call features. We have divided the features in different groups such as personal permissions, personal API calls, system permissions, system API calls, device permissions, device API calls, Network permissions, Network API calls, OS permissions, and OS API calls. Permissions and the API call are the most significant features in android malware detection. Permissions are obtained from the manifest files and the global set of permissions is generated according to the Algorithm 2. The graph showing the percentage of application requesting permissions vs the number of permissions is plotted. Fig. 5 shows that the malware data set uses significant number of permissions whereas the benign data set shows low number of permissions. Moreover, 20–45 permissions are mostly used in malware compared to benign ware and multiple spikes are shown in this region in the figure.

Fig. 5 is generated based on the global permission set. Here we have illustrated the difference of malware and benign ware while using the permission set. Malware have used higher number of permissions than the benign ware. Most of the benign ware used less than 20 permissions.

The API call are extracted from the apk files and the global set of API calls has been generated according to the Algorithm 2. We have compared the characteristics of using API calls in both malware and benign ware. Fig. 6 considers the global set of API calls and illustrates the behavioral difference of malware and benign ware in using the API calls. A large number of benign wares use few API calls. The following graph (Fig. 6) has shown that malware uses more API calls compared to the Benign ware. Highest number of API used in benign ware is only 25 whereas 85 API calls used in malware.

Since the features are selected from the global set, we have regenerated the graphs using the selected number of features. Since, we have selected the features using Boruta and grouped them in six different groups, six more graphs have been obtained with the selected features. Fig. 7 describes the comparative analysis of the usage of requesting permissions of malware and benign ware with the selected permission set. It has been observed that few numbers of benign ware use high number of permissions. Fig. 8 is generated considering the selected API call and shows a visible difference in the nature of malware and benign ware. Most of the benign ware use a very few numbers of API calls. In Fig. 8, highest number of API call used by the benign ware is 25 whereas it is 85 for the malware.

The Figs. 7 and 8 shows that the malware uses more permissions and API calls compared to the benign ware. From these graphs, we have found that both permissions and API calls play vital role in Android mobile malware detection process. As a result, we considered both permissions and API calls as features.

Moreover, we have analyzed the behavior of malware while using the API calls. We have grouped the API calls into four groups: system, personal, network and OS. A significant difference has been observed. Fig. 9 has represented the usage the network related selected API calls. Most of the malware uses higher number of network related API calls whereas 40% of benign ware uses only 2–3 network related API calls (Fig. 9). Fig. 10 illustrates the nature of using OS related API calls of malware and benign ware and a significant difference has been observed here. The highest number of OS related API calls used in benign ware is 13 whereas it is 26 in case of malware. The most significant differences have been shown in network and system API call, the highest number of Network related API calls in benign ware is 5 where as it is 30 for malware. Similarly, the highest number of system related API calls in benign ware is 6 where as it is 26 for malware. From these graphs, we can say that most of the benign ware uses a smaller number of API calls compared to the malware. The following figures (Figs. 9, 10, 11, 12) shows the wide differences in using the network, OS, personal and system related API calls in two data sets-malware and benign ware. Fig. 11 expresses

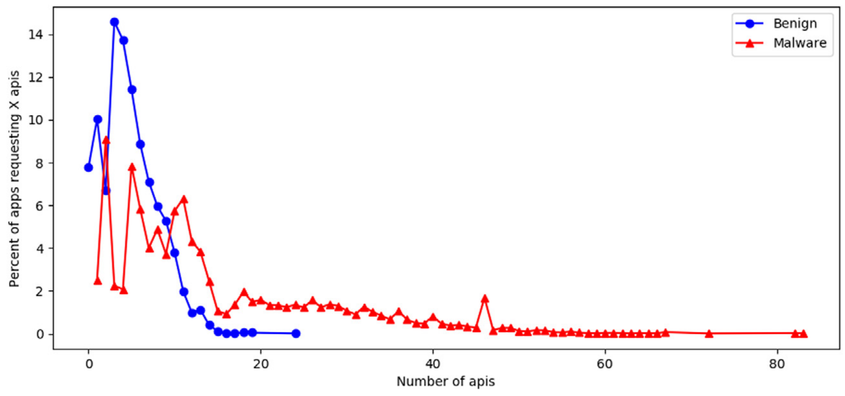


Fig. 6. Percentage of application requesting API calls (using global set of API calls).

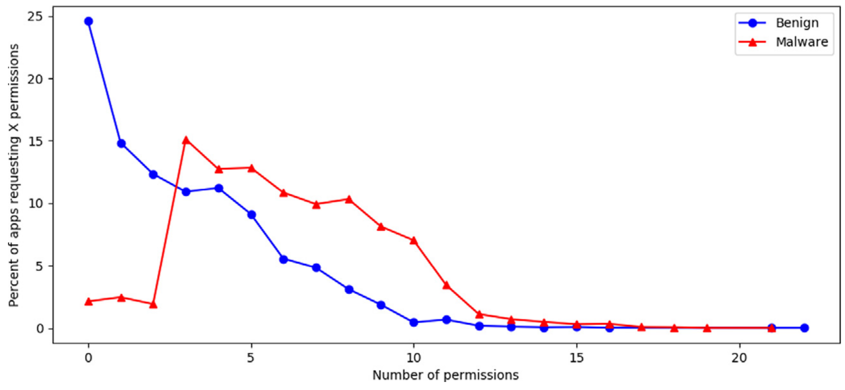


Fig. 7. Percentage of application requesting permissions (using selected permissions).

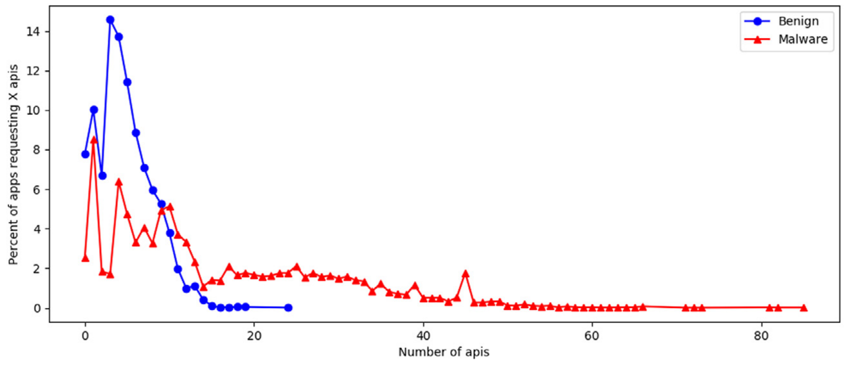


Fig. 8. Percentage of application requesting API calls (using selected API calls).

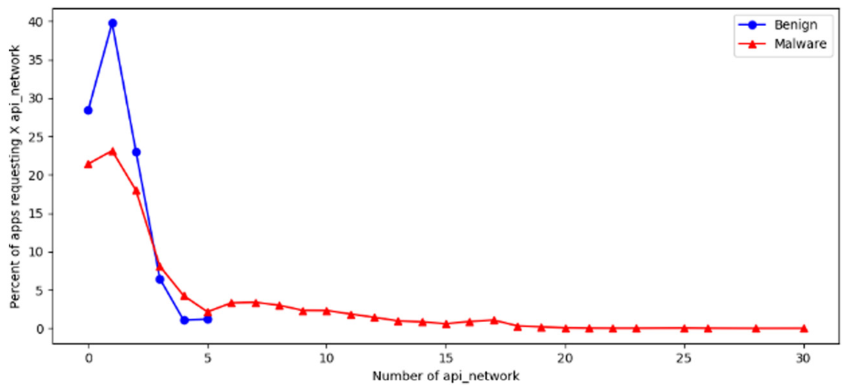


Fig. 9. Percentage of application requesting network related API calls (using selected API calls).

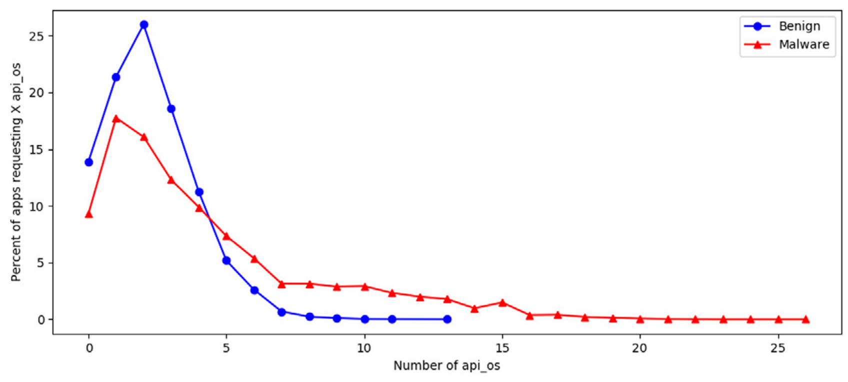


Fig. 10. Percentage of application requesting OS related API calls (using selected API calls).

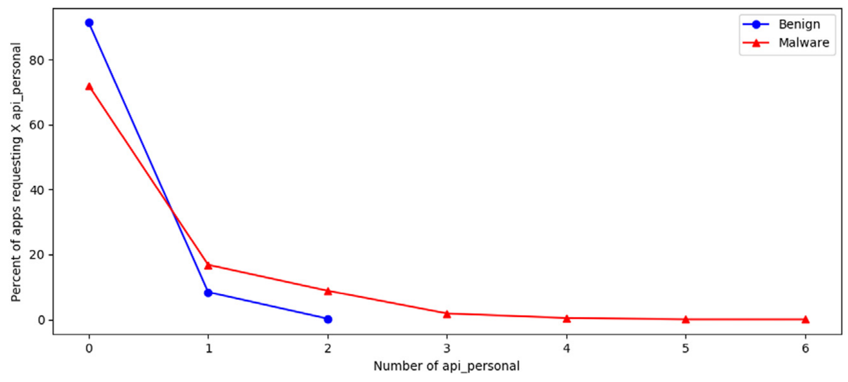


Fig. 11. Percentage of application requesting personal information related API calls (using selected API calls).

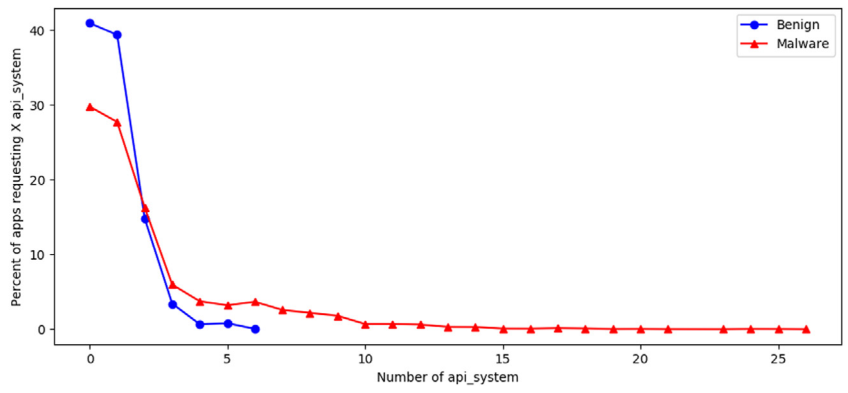


Fig. 12. Percentage of application requesting system related API calls (using selected API calls).

how does the malware differ from benign ware in using the personal information related API calls. Most of the benign ware and malware uses relatively low number of personal information related API call. Still a clear difference is observed-malware uses relatively higher number of personal related API calls. Fig. 12 is associated with the usage of system related API calls by the malware and the benign ware. Benign ware uses 3–4 system related API calls. The highest number of system related API that are used by the malware is 26. A mentionable dissimilarity is observed here in Fig. 12.

4.5. Integration of supervised and unsupervised method

As we have presented in Fig. 3(b), the feature vector generation in previous section, we will incorporate the distance as the knowledge from the unsupervised clustering in the feature vector. We will use the knowledge of unlabeled data in two ways — clustering and pseudo labeling. In first step we will combine both labeled and unlabeled data and apply a clustering library to find clusters. Distance from each cluster will be added as new features in the feature vector matrix to train our model. In second step this train model will be used to predict the label of unlabeled dataset. This newly labeled data set will be combined with the original labeled dataset. We will retrain our model using this combined data set.

4.5.1. Supervised learning

In our detection model, at first the supervised learning will be applied to do the training using the labeled data and the model is then tested using test data set. The detection model used the selected features to do the training of the model. To train the model, we have used different types of classifier such as the tree-based classifiers (Random Forest [30], j48, extratree, Decision tree), k neighbors, Gradient Boosting etc. We have also considered the support vector machine [51].

4.5.2. Clustering

Unsupervised method uses the measured distance which will be determined using term frequency–inverse document frequency (TF*IDF). For calculating this, we have split features set into several groups and calculated the frequency for all groups. TF–IDF will be calculated using the group frequency. The resulted matrix will be like following Fig. 13.

TF–IDF will be used in K-mean algorithm to determine the clusters and the cluster centers. Cosine similarity from each cluster centers is considered as the distance and will be inserted as a new feature in the feature vector matrix. Thus, two distance columns are used to inject in the feature vector which represent the incorporation of knowledge of the unsupervised learning to supervised learning. The incorporation of the knowledge of unsupervised learning is illustrated in Fig. 14.

Now we will train the model with the resulted dataset. In our next step of semi supervised approach, this model is used to predict the label for unlabeled data.

4.5.3. Pseudo labeling

We used an iterative process to label the unlabeled data. The unlabeled dataset is split into ten small data set. We used our trained model to predict the label of first portion of unlabeled data set and merge this result to retrain the model. The process of iterative prediction and retrain continues until all ten data set is used.

We predicted the label of the unlabeled data which is known as Pseudo Labeling. In our algorithm, we have represented the steps of supervised learning, pseudo leveling and incorporation of the knowledge from clustering as follows.

5. Experimental results

5.1. Performance matrices

We have considered the True Positive rate (TPR), False Positive rate (FPR), Accuracy, Recall, Precision, F1_score and AUC_ROC curve as the performance matrices. Since ROC curve is the most important tool to proof the efficiency and reliability of a model [52], we have considered the ROC curve as a performance metric. As accuracy alone cannot define the model, ROC_AUC Curve defines capability of distinguishing between classes. Higher ROC_AUC curve score is more desirable. The details of finding the area under the ROC curve is discussed in [52] and we have followed that. We have determined the ROC_AUC metric of our model and represented in the result section. If, TP is the number of accurately identified benign apps, FN is the number of wrongly identified benign apps, FP is the number of wrongly identified malware and TN is the number of correctly identified malware then we can describe the followings:

$$TPR = TP/(TP + FN) \tag{19}$$

$$FPR = FP/(TN + FP) \tag{20}$$

$$ACC = (TP + TN)/(TP + TN + FP + FN) \tag{21}$$

$$Precision = TP/(TP + FP) \tag{22}$$

$$Recall = TP/(TP + FN) \tag{23}$$

$$F - measure = 2 * (Precision * Recall)/(Precision + Recall) \tag{24}$$

5.2. Performance comparison criteria

Data set described in Table 4 has been considered here to evaluate the performance of the models. We have compared the performance of Model-1 by varying the node numbers in hidden layers. Moreover, we varied the epoch numbers. To compare the performance of different node arrangements in model-1, we measured the accuracy, loss, recall, precision, f1score, Area under ROC curve, TPR, TNR, FPR and FNR. Among them, we have created the performance graphs using accuracy and loss. These graphs show the learning process of model-1 considering accuracy and loss after every epoch. Moreover, a comparison of supervised method and semi-supervised model have been illustrated in the result section. TPR, FPR, recall, precision, f1 score, ROC score and accuracy have been calculated to compare these supervised and semi-supervised models. A chart showing the comparison of the achieved accuracy level has been constructed. Additionally, ROC curves are generated to show the comparison of the performance of semi-supervised model.

5.3. Simulation environment

Our experiment was conducted using Intel® Xenon® CPU E5-2630V4 2.20 GHz (2 processors), 128 GB physical memory and eight NVIDIA GFORCE GTX 1080Ti GPU cards. Deep learning-based model is developed using TensorFlow [53]. To get the full benefit of parallel processing and NVIDIA CUDA based GPU's our experiment used the python based tensorflow-gpu 1.14 library. We have considered 10-cross fold validation to train and test the models. TensorFlow used Relu and softmax [54] as the activation functions and Adam [55] as optimizer. Sigmoid cross entropy loss function [56] is considered to measure the loss. Semi-supervised model is developed using the python script and GPU based parallel processing is also considered here.

5.4. Experiment outcomes

In the very first stage, we have developed our detection model using Deep learning method in TensorFlow [53]. The global set of features (Algorithm 2) are considered here. In deep learning-based model, we need to work with multiple layers with a large number of nodes in each layer. A large number of computations are needed to be executed. To address this issue, we have considered GPU based parallel processing. In our model we have considered the GPU supported TensorFlow. In our model we have considered 2 hidden layers and set the epoch number as 500. We have varied the number of nodes in each layer. The different architectures of node arrangements have been deployed here. The highest accuracy level 99.024% has been achieved using equal number of nodes 1024 in each layer. The detailed performance result has been given below with 500 epochs:

In Table 5, we have represented the performance of the detection model-1 using deep learning considering eight combinations of nodes in the hidden layers-L32_L32 (32 nodes in each layer), L32_L64 (32 nodes in layer 1 and 64 nodes in layer 2), L64_L32 (64 nodes in layer 1 and 32 nodes in layer 2), L64_L64(64 nodes in each layer), L512_L512 (512 nodes in each layer), L1024_L512 (1024 nodes in layer 1 and 512 nodes in layer 2), L1024_L1024 (1024 nodes in each layer) and L1024_L64(1024 nodes in layer 1 and 64 nodes in layer 2). Performance metrics (accuracy, loss, recall, precession, f1_score, Area under ROC curve, TPR, TNR, FPR and FNR) are calculated for these eight combinations. L1024_L1024(1024 nodes in each layer) shows the best accuracy level (99.024%). ROC score is also very close to 1 (0.9848).

	Group 1	Group 2	….	Group N	TF * IDF 1	TF * IDF 2	….	TF * IDF N
Apk 1	2	4		2	…..		…..	
Apk 2	1	3		5	…….		…….	
……					……		……	
Apk 3	4	2		0	……..		……..	

Fig. 13. TF*IDF measurement.

	Permission 1	Permission 2	…	Permission N	API call 1	API call 2	…	API call N	**Distance 1**	**Distance 2**	Is malware/ not?
App 1	1	0	….	0	1	1	….	1			0
App 2	1	1	….	1	0	1	……	1			1
App 3	0	1	….	0	1	0	……	1			1
……	…..	……	….	……	…..	…..	……	…..			…….
…….	…..	…….	….	…….	…..	…..	…	…..			……..
App N	1	0	…	1	1	0	….	1			1

Fig. 14. Incorporation of knowledge from unsupervised learning in feature vector.

Algorithm 4: Semi-supervised learning: incorporating the unsupervised learning knowledge to supervised learning

```
1.  $input \leftarrow DL(F_1F_2, F_3 \dots\dots\dots\dots F_m)$ {Labeled data set with m number of global features}
2.  $input \leftarrow UDL(F_1F_2, F_3 \dots\dots\dots\dots F_m)$ {Unlabeled data set with m number of global features}
3.  Output $Acc_1, Acc_2, Acc_3$
4.  Begin
5.  Train and test the model using 10-fold cross validation for $DL$ and Calculate accuracy, $Acc_1$.
    {supervised method using global features set}
6.  Do the feature selection using Boruta wrapper model
7.  Reconstruct feature vector $DL$ and form $DL_1$ using selected features
8.  Train and test the model using 10-fold cross validation for $DL_1$ and Calculate accuracy, $Acc_2$
    {supervised method using selected features set}
9.  Reconstruct feature vector $UDL$ and form $UDL_1$ using selected features
10. Train the model using $DL_1$
11. Predict the label of unlabeled data and by inserting the label of the unlabeled data, construct a new
    data set $DL_2$ {pseudo labeling}
12. Construct a new data set $DL_3$ by doing union of $DL_1$ and $DL_2$.
13. Remove the label of the dataset $DL_3$ and construct a new unlabeled data set $UDL_2$
14. Find the clusters using K-means algorithm using $UDL_2$. {unsupervised method using selected
    features set}
15. Find the cluster centers.
16. Compute distance of instance from the cluster center.
17. Inject the distance in the feature vector $DL_3$ and form a new feature vector $DL_4$. {combine the
    unsupervised learning to supervised learning}
18. Retrain and test the classifier using 10-fold cross validation and calculate the accuracy, $Acc_3$ for
    $DL_4$ {semi supervised method}
19. END
```

The detailed results (Table 5) of deep learning-based model have shown that equal number of nodes in each layer provides the hugest accuracy level. As we have also varied the epoch number, we have observed that the higher epoch provides higher accuracy with low loss.

We have also implemented the semi-supervised model. The second phase of our malware detection model — semi supervised method (Algorithms 1, 2, 3 and 4) is executed in python. At the very first level, we have implemented the supervised model including different types of classifier and tested the model using the data set mentioned in Table 2. The detailed result of supervised model has been shown in Table 6. In the supervised method, the Random forest and ExtraTree show comparatively better accuracy level than others. SVM shows 92.44% of accuracy level.

In Table 6, we have represented the performance of the supervised model using five different classifiers named Random Forest, ExtraTree, Decision tree, SVM and XRGB. True positive rate, false positive rate, precision, recall, F1 score, Roc score and accuracy of each supervised classifying algorithm is evaluated. The highest

Table 5
Performance of deep learning model based on first proposed detection model.

	L32_L32	L32_L64	L64_L32	L64_L64	L512_L512	L1024_L512	L1024_L1024	L1024_L64
Accuracy	98.753	98.897	99.014	98.881	98.993	98.990	**99.024**	98.993
Loss	0.0544	0.0499	0.0479	0.0534	0.0484	0.0471	0.0473	0.0470
Recall	0.7812	0.8034	0.8352	0.7960	0.8230	0.8273	0.8329	0.8257
Precession	0.9177	0.9319	0.9297	0.9352	0.9361	0.9309	0.9338	0.9339
f1 score	0.8436	0.8624	0.8795	0.8596	0.8756	0.8759	0.8801	0.8758
AUC	0.9807	0.9833	0.9850	0.9833	0.9849	0.9856	0.9848	0.9866
TPR	0.9902	0.9912	0.9926	0.9908	0.9920	0.9922	0.9925	0.9921
TNR	0.9172	0.9312	0.9289	0.9347	0.9355	0.9307	0.9334	0.9331
FPR	0.0827	0.0687	0.0710	0.0652	0.0644	0.0692	0.0665	0.0668
FNR	0.0097	0.0087	0.0073	0.0091	0.0079	0.0077	0.0074	0.0078

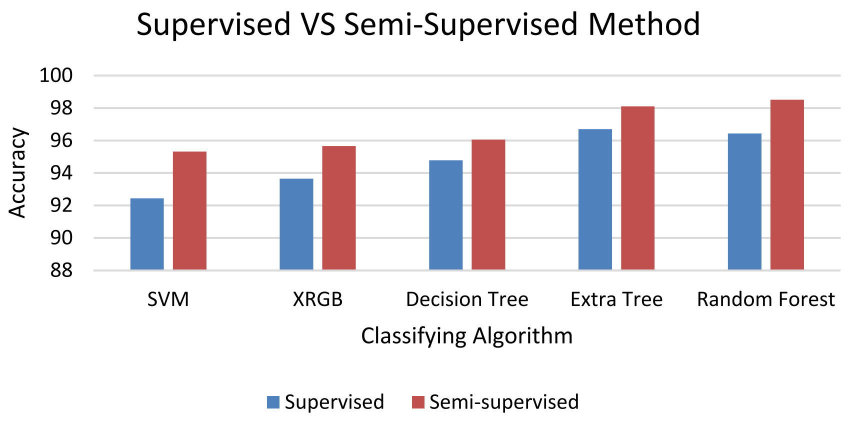


Fig. 15. Comparison of supervised and semi-supervised method.

accuracy level (96.7%) is achieved by the ExtraTree algorithm and SVM achieves the lowest accuracy level (92.44%).

Then semi-supervised model has been developed using the same set of classifiers as the supervised model. The semi-supervised model has shown better performance than the supervised method. The Table 7 represents the detailed performance of semi-supervised model. Random Forest and ExtraTree classifier have shown approximately 99% of accuracy level. All of the classifiers have improved the achieved accuracy level using the semi supervised approach and the FPR decreases significantly.

In Table 7, we have represented the performance of the semi-supervised model using five different classifiers named Random Forest, ExtraTree, Decision tree, SVM and XRGB. The same performance metrics are used here as the supervised method. The highest accuracy level (98.51%) is achieved by the random forest algorithm and SVM achieves the lowest accuracy level (95.31%). The ROC score for Random Forest and ExtraTree is 0.996 which is very close to 1.

In Fig. 15, we have represented the performance based on achieved accuracy level of the supervised and semi-supervised method. Semi-supervised SVM has shown 2.87% of improvement in the achieved accuracy level than that of supervised model. Similarly, semi-supervised XRGB, Decision tree, Extra tree and Random forest accomplished 2.02%, 1.27%, 1.4% and 2.08% of increment in the accuracy level respectively. Here, we have noticed the clear improvements in terms of accuracy level using semi-supervised method.

As we have implemented the supervised method, semi-supervised method and deep learning-based model, we have compared the results of our model with most recent and related research works. The comparison is illustrated in the following Table 8.

In Table 8, we have put the related works, publishing years and the achieved accuracy levels. We have considered the recent works as well.

From the comparison table, we have observed that he deep learning method has achieved 99.024% of accuracy level which is the highest one among all other methods. Semi-supervised based model has incorporated the opportunity to integrate the knowledge of unlabeled data and unsupervised learning method. This will help to detect new and unknown malware and will be helpful to detect new malware. Semi supervised method has achieved 98.51% of accuracy level which is also satisfactory one.

Other research works using Deep learning-based approaches [15,18,20–24] have achieved lower accuracy level than our model. Most of these research works are from the year 2018 and 2019. The superiority of our proposed approaches is that, we have combined the knowledge from new unlabeled applications which we get from the unsupervised learning and this knowledge has effectively enhance the accuracy level. This integration provided positive impact in detection process in deep learning-based model-1 and the achieved accuracy level proves that (Table 8). Moreover, we have considered higher epoch number and different arrangements of nodes to find out the optimized result. This adaptive framework can cope up with the frequent changes of the nature of the malware, the intrinsic nature of the malware can be extracted, and this knowledge is helpful to ensure zero-day detection. Semi-supervised model-2 uses the features from labeled and unlabeled data set, this also increases the possibility to detect the malware. Additionally, data set from four different sources are combined to get the labeled and unlabeled data. The behavioral difference of malware using API call and permissions are also examined.

Our detection model is scalable since we can feed as much data as we want. The availability of labeled data is not a concern here. Since no manual labeling is needed here and we can use unlabeled data, this property of our detection model makes it unique. Moreover, the automatic update of malware database is also implemented here.

Since Accuracy level alone cannot describe the performance of the model, we have considered the ROC curve and also achieved

Table 6
Performance of supervised model based on second proposed detection model.

	TPR	FPR	Precision	Recall	F1 score	ROC score	Accuracy
Random Forest	0.961	0.033	0.964	0.964	0.964	0.993	96.43%
Extratree	0.958	0.025	0.967	0.967	0.967	0.992	96.7%
Decision	0.944	0.050	0.948	0.948	0.948	0.950	94.78%
SVM	0.909	0.063	0.925	0.924	0.925	0.972	92.44%
XGB	0.920	0.053	0.935	0.935	0.935	0.981	93.64%

Table 7
Performance of semi-supervised model based on second proposed detection model.

	TPR	FPR	Precision	Recall	F1 score	ROC score	Accuracy
Random Forest	0.985	0.014	0.985	0.985	0.985	0.996	98.51%
Extratree	0.968	0.013	0.981	0.981	0.981	0.996	98.10%
Decision	0.947	0.033	0.960	0.960	0.960	0.953	96.05%
SVM	0.928	0.034	0.953	0.953	0.953	0.986	95.31%
XRGB	0.927	0.028	0.957	0.957	0.957	0.989	95.66%

Table 8
Comparison with other related research works.

Ref.	Year	Accuracy level
A novel approach for mobile malware classification and detection in Android systems [13]	2019	97.85%
Deep Learning in Drebin: Android malware Image Texture Median Filter Analysis and Detection [15]	2019	95.43%
Malware Detection Using Machine Learning Algorithms and Reverse Engineering of Android Java Code [17]	2019	80.7%
MalDozer: Automatic framework for android malware detection using deep learning [18]	2018	90%
DroidDet: effective and robust detection of android malware using static analysis along with rotation forest model [19]	2018	88.26%
Deep android malware detection [20]	2017	98%
Droid-Sec: Deep learning in android malware detection [21]	2017	96.5%
A multimodal deep learning method for Android malware detection using various features [22]	2018	98%
A novel android malware detection approach based on convolutional neural network [23]	2018	97.45%
Droiddetector: android malware characterization and detection using deep learning [24]	2016	96.76%
Fine-grained Android Malware detection based on Deep learning [6]	2018	97.16%
Hybrid Android Malware Detection by Combining Supervised and Unsupervised Learning [7]	2018	91.98%
Significant Permission Identification for Machine Learning Based Android Malware Detection [4]	2018	93.62%
FgDetector: Fine-grained Android Malware Detection [3]	2017	98.6%
Iccdetector: Icc-based malware detection on android [34]	2016	97.4%
Machine learning based hybrid behavior models for Android malware analysis [33]	2015	96.69%
Android Malware Detection Using Parallel Machine Learning Classifiers [8]	2014	98.6%
The analysis of feature selection methods and Classification algorithms in Permission based Android Malware Detection [29]	2014	94.90%
DREBIN: Effective and Explainable Detection of Android Malware in Your Pocket [31]	2014	93.4%
MADAM: a multi-level anomaly detector for android malware [25]	2012	96.9%
DroidMat: Android Malware Detection through Manifest and API call Tracing [26]	2012	97.87%
Using Probabilistic Generative Models for Ranking Risks of Android Apps [28]	2012	88.2%
Android: From reversing to Decompilation [27]	2011	93.04%
Our Detection Model-1 using Deep Learning Approach	2019	**99.024%**
Our Detection Model-2 using Semi-supervised Approach	2019	98.51%

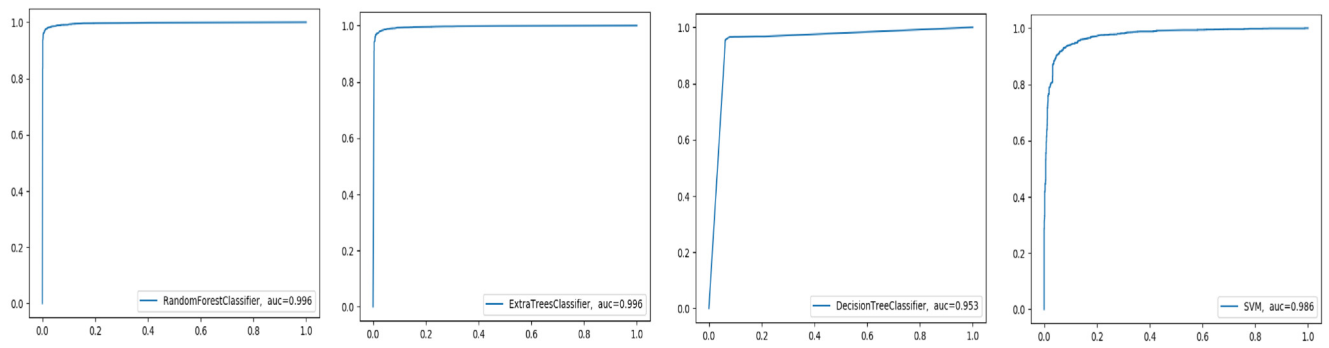


Fig. 16. Area under ROC curves of semi-supervised method.

excellent performance level using semi-supervised method. The ROC curves show nearly 0.996 (very close to 1) area under ROC score. Fig. 16 shows the ROC analysis of the semi-supervised model. In Fig. 16, we have accumulated 4 ROC curves of semi-supervised Random Forest, Extratree, Decision tree and SVM. Among them, Random Forest and ExtraTree have achieved 0.996 and 0.995 as ROC scores. Semi-supervised Decision tree and SVM show also satisfactory ROC score.

Moreover, for our deep learning-based model, we have plotted the performance curves defining Accuracy and Loss. We have considered sigmoid cross entropy loss function [56] to define the Loss. Accuracy and Loss have been measured after each epoch and plotted in the graph. Accuracy has been increased with the increasing number of the epoch and loss has been decreased with the increasing number of epochs. The Adam optimizer [55] has been used to optimize the accuracy and minimize the Loss. The

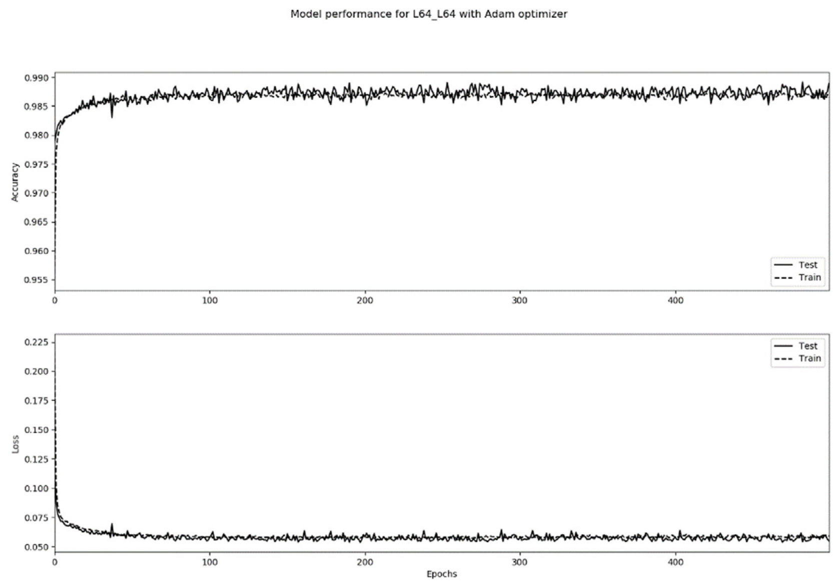


Fig. 17. Accuracy VS epoch number and Loss VS epoch number for the arrangement with 64 nodes in each layer (Layer 1 and Layer 2).

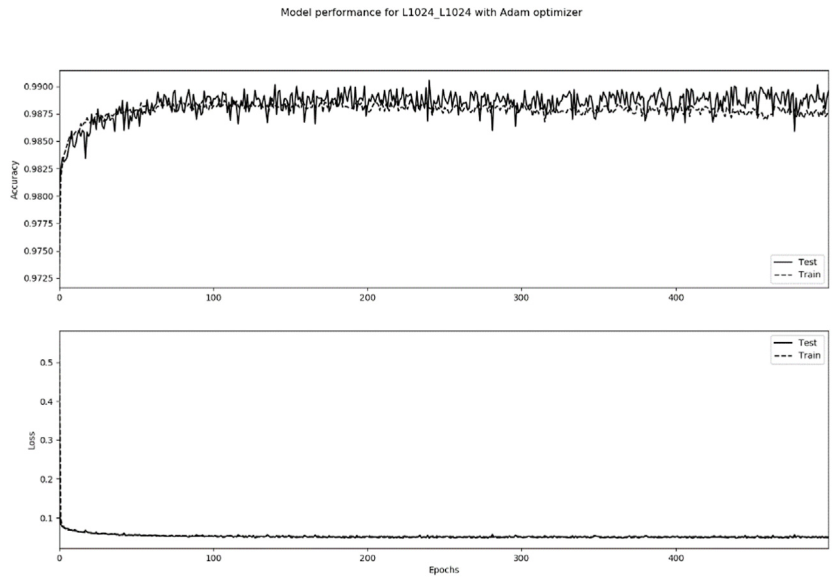


Fig. 18. Accuracy VS epoch number and Loss VS epoch number for the arrangement with 1024 nodes in each layer (Layer 1 and Layer 2).

following performance graphs (Figs. 17 and 18) shows the nature of accuracy and loss according to the epoch number. Since we have varied the arrangements of the nodes and number of the nodes, we obtained a large number of performance graphs. Fig. 17 represents the arrangement with equal number of nodes in Layer 1 and Layer 2 with 64 nodes in each layer. Additionally, Fig. 18 shows the performance graphs of the arrangement with 1024 nodes in layer 1 and Layer 2. Fig. 17 consists of two graphs — the upper one describes the achieved accuracy versus the epoch numbers and the lower one represents the loss versus the epoch number. With the increment of the epoch number, the accuracy level increases at the very first stage then it fluctuates in a very tiny range. The loss decreases with the increase of the epoch numbers. Similarly, in Fig. 18 we have noticed that the loss is reduced, and accuracy is increased with the increase of epoch number.

6. Conclusion and future work

This paper proposes a novel adaptive framework for mobile malware detection system to ensure zero-day protection. Two different detection models are developed using unsupervised and semi-supervised learning for our adaptive framework by implementing the deep learning based model and combining the unsupervised and supervised techniques into a semi-supervised model. The key novelty of our semi-supervised models is that they can extract the hidden intrinsic patterns from new malware and can integrate that knowledge to the supervised engine to detect new malware and ensure zero-day protection. Thus, proposed framework is adaptive to the dynamic behavior of malicious apps in the android market. Deep learning technique in our framework uses abstraction and non-linear transformation of raw input data to higher layer abstraction and discovers hidden patterns automatically from unlabeled malicious apps. While semi-supervised approach in the proposed framework extracts the hidden abuse patterns by unsupervised clustering and combine that to the supervised classifier using a Euclidean distance metric. However, deep learning's performance depends on the settings of the hyper-parameters such as the size of mini batch, number of epochs, and number of hidden layers. In our adaptive framework, we overcome this bottleneck by using variable structure of deep learning. In the second proposed model of semi-supervised approach, clustering is used which is a much simpler training process compared to varying structure of deep learning of the proposed first model. Comprehensive evaluations of proposed approaches by using a real malware test-bed and data set show that deep learning based proposed detection model achieves the best performance, an accuracy over 99%, among all different detection engines including existing supervised models and proposed second model based on unsupervised learning which achieves 98.51%. This justifies the effectiveness and significance of our proposed approaches. In the future, we will consider combination of dynamic and static features to ensure 100% accuracy for zero-day protection.

Acknowledgment

The authors are grateful to King Saud University, Riyadh, Saudi Arabia for funding this work through Researchers Supporting Project number RSP-2019/18.

Declaration of competing interest

No author associated with this paper has disclosed any potential or pertinent conflicts which may be perceived to have impending conflict with this work. For full disclosure statements refer to https://doi.org/10.1016/j.asoc.2020.106089.

References

[1] https://www.statista.com/statistics/266136/global-market-share-held-by-smartphone-operatingsystems/. (Access Date: 30.11.2018).
[2] Ping Yan, Zheng Yan, A survey on dynamic mobile malware detection, Softw. Qual. J. 26 (3) (2018) 891–919.
[3] D. Li, Z. Wang, L. Li, Z. Wang, Y. Wang, Y. Xue, FgDetector: Fine-grained Android malware detection, in: Data Science in Cyberspace, DSC, 2017 IEEE Second International Conference on 2017 Jun 26, IEEE, 2017, pp. 311–318.
[4] J. Li, L. Sun, Q. Yan, Z. Li, W. Srisa-an, H. Ye, Significant permission identification for machine learning based Android malware detection, IEEE Trans. Ind. Inf. (2018).
[5] M. Odusami, O. Abayomi-Alli, S. Misra, O. Shobayo, R. Damasevicius, R. Maskeliunas, Android malware detection: A survey, in: International Conference on Applied Informatics 2018 Nov 1, Springer, Cham., 2018, pp. 255–266.
[6] D. Li, Z. Wang, Y. Xue, Fine-grained Android malware detection based on deep learning, in: 2018 IEEE Conference on Communications and Network Security, CNS 2018 May 30, IEEE, 2018, pp. 1–2.
[7] A. Arora, S.K. Peddoju, V. Chouhan, A. Chaudhary, Poster: Hybrid Android malware detection by combining supervised and unsupervised learning, in: Proceedings of the 24th Annual International Conference on Mobile Computing and Networking 2018 Oct 15, ACM, 2018, pp. 798–800.
[8] S.Y. Yerima, S. Sezer, I. Muttik, Android malware detection using parallel machine learning classifiers, in: 2014 Eighth International Conference on Next Generation Mobile Apps, Services and Technologies 2014 Sep 10, IEEE, 2014, pp. 37–42.
[9] L. Aneja, S. Babbar, Research trends in malware detection on Android devices, in: International Conference on Recent Developments in Science, Engineering and Technology 2017 Oct 13, Springer, Singapore, 2017, pp. 629–642.
[10] S.Y. Yerima, S. Sezer, I. Muttik, High accuracy android malware detection using ensemble learning, IET Inf. Secur. 9 (6) (2015) 313–320.
[11] S.B. Kotsiantis, I. Zaharakis, P. Pintelas, Supervised machine learning: A review of classification techniques, Emerg. Artif. Intell. Appl. Comput. Eng. 160 (2007) 3–24.
[12] M. Grace, Y. Zhou, Q. Zhang, S. Zou, X. Jiang, Riskranker: scalable and accurate zero-day android malware detection, in: Proceedings of the 10th International Conference on Mobile Systems, Applications, and Services 2012 Jun 25, ACM, 2012, pp. 281–294.
[13] Q. Zhou, F. Feng, Z. Shen, R. Zhou, M.Y. Hsieh, K.C. Li, A novel approach for mobile malware classification and detection in Android systems, Multimedia Tools Appl. 78 (3) (2019) 3529–3552.
[14] W. Wang, M. Zhao, J. Wang, Effective android malware detection with a hybrid model based on deep autoencoder and convolutional neural network, J. Ambient Intell. Humaniz. Comput. 10 (8) (2019) 3035–3043.
[15] L. Shi-qi, N. Bo, J. Ping, T. Sheng-wei, Y. Long, W. Rui-jin, Deep learning in Drebin: Android malware image texture median filter analysis and detection, KSII Trans. Internet Inf. Syst. 13 (7) (2019) 3654–3670.
[16] Y.S. Hamed, S.N. AbdulKader, M.S. Mostafa, Mobile malware detection: A survey, Int. J. Comput. Sci. Inf. Secur. 17 (1) (2019).
[17] M. Kedziora, P. Gawin, M. Szczepanik, I. Jozwiak, Malware detection using machine learning algorithms and reverse engineering of Android Java code, Int. J. Netw. Secur. Appl. (2019) 11.
[18] E.B. Karbab, M. Debbabi, A. Derhab, D. Mouheb, MalDozer: Automatic framework for android malware detection using deep learning, Digit. Investig. 24 (2018) S48–59.
[19] H.J. Zhu, Z.H. You, Z.X. Zhu, W.L. Shi, X. Chen, L. Cheng, DroidDet: effective and robust detection of android malware using static analysis along with rotation forest model, Neurocomputing. 272 (2018) 638–646.
[20] N. McLaughlin, J. Martinez del Rincon, B. Kang, S. Yerima, P. Miller, S. Sezer, Y. Safaei, E. Trickel, Z. Zhao, A. Doupé, G. Joon Ahn, Deep android malware detection, in: Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy 2017 Mar 22, ACM, 2017, pp. 301–308.
[21] Z. Yuan, Y. Lu, Z. Wang, Y. Xue, Droid-sec: deep learning in android malware detection, in: ACM SIGCOMM Computer Communication Review 2014 Aug 17, Vol. 44 No. (4), ACM, 2014, pp. 371–372.
[22] T. Kim, B. Kang, M. Rho, S. Sezer, E.G. Im, A multimodal deep learning method for Android malware detection using various features, IEEE Trans. Inf. Forensics Secur. 14 (3) (2018) 773–788.
[23] Y. Zhang, Y. Yang, X. Wang, A novel android malware detection approach based on convolutional neural network, in: Proceedings of the 2nd International Conference on Cryptography, Security and Privacy 2018 Mar 16, ACM, 2018, pp. 144–149.
[24] Z. Yuan, Y. Lu, Y. Xue, Droiddetector: android malware characterization and detection using deep learning, Tsinghua Sci. Technol. 21 (1) (2016) 114–123.
[25] G. Dini, F. Martinelli, A. Saracino, D. Sgandurra, MADAM: a multi-level anomaly detector for android malware, in: International Conference on Mathematical Methods, Models, and Architectures for Computer Network Security 2012 Oct 17, Springer, Berlin, Heidelberg, 2012, pp. 240–253.
[26] D.J. Wu, C.H. Mao, T.E. Wei, H.M. Lee, K.P. Wu, Droidmat: Android malware detection through manifest and api calls tracing, in: Information Security, Asia JCIS, 2012 Seventh Asia Joint Conference on 2012 Aug 9, IEEE, 2012, pp. 62–69.
[27] A. Desnos, G. Gueguen, Android: From reversing to decompilation, in: Proc. of Black Hat Abu Dhabi. 2011 Dec, 2011, pp. 77–101.
[28] H. Peng, C. Gates, B. Sarma, N. Li, Y. Qi, R. Potharaju, C. Nita-Rotaru, I. Molloy, Using probabilistic generative models for ranking risks of android apps, in: Proceedings of the 2012 ACM Conference on Computer and Communications Security 2012 Oct 16, ACM, 2012, pp. 241–252.
[29] U. Pehlivan, N. Baltaci, C. Acartürk, N. Baykal, The analysis of feature selection methods and classification algorithms in permission-based Android malware detection, in: Computational Intelligence in Cyber Security, CICS, 2014 IEEE Symposium on 2014 Dec 9, IEEE, 2014, pp. 1–8.
[30] L. Breiman, Random forests, Mach. Learn. 45 (1) (2001) 5–32.
[31] D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, K. Rieck, C.E. Siemens, DREBIN: Effective and explainable detection of android malware in your pocket. in: InNdss 2014 Feb 23, vol. 14, 2014, pp. 23–26.
[32] X. Liu, J. Liu, A two-layered permission-based android malware detection scheme, in: 2014 2nd IEEE International Conference on Mobile Cloud Computing, Services, and Engineering 2014 Apr 8, IEEE, 2014, pp. 142–148.
[33] H.Y. Chuang, S.D. Wang, Machine learning based hybrid behavior models for Android malware analysis, in: 2015 IEEE International Conference on Software Quality, Reliability and Security 2015 Aug 3, IEEE, 2015, pp. 201–206.
[34] K. Xu, Y. Li, R.H. Deng, Iccdetector: Icc-based malware detection on android, IEEE Trans. Inf. Forensics Secur. 11 (6) (2016) 1252–1264.
[35] K. Wagstaff, C. Cardie, S. Rogers, S. Schrödl, Constrained k-means clustering with background knowledge, in: InIcml 2001 Jun 28, vol. 1, 2001, pp. 577–584.
[36] A. Singh, A. Yadav, A. Rana, K-means with three different distance metrics, Int. J. Comput. Appl. 67 (10) (2013).
[37] W. Zhang, T. Yoshida, X. Tang, A comparative study of TF* IDF, LSI and multi-words for text classification, Expert Syst. Appl. 38 (3) (2011) 2758–2765.
[38] S. Tata, J.M. Patel, Estimating the selectivity of TF-IDF based cosine similarity predicates, ACM Sigmod Rec. 36 (2) (2007) 7–12.

[39] J.H. Paik, A novel TF-IDF weighting scheme for effective ranking, in: Proceedings of the 36th International ACM SIGIR Conference on Research and Development in Information Retrieval 2013 Jul 28, ACM, 2013, pp. 343–352.

[40] A. Huang, Similarity measures for text document clustering, in: Proceedings of the sixth New Zealand computer science research student conference, NZCSRSC2008, Christchurch, New Zealand 2008 Apr 14, vol. 4, 2008, pp. 9-56.

[41] Yann LeCun, Yoshua Bengio, Geoffrey Hinton, Deep learning, Nature 521 (2015) 436–444.

[42] Robert Mitchell, Ing-Ray Chen, Behavior rule specification-based intrusion detection for safety critical medical cyber physical systems, IEEE Trans. Dependable Secure Comput. 12 (1) (2015).

[43] C.A. Bouman, M. Shapiro, G.W. Cook, C.B. Atkins, H. Cheng, J.G. Dy, S. Borman, Cluster: An unsupervised algorithm for modeling Gaussian mixtures, in: Software Package, 2005, http://www.ece.purdue.edu/~bouman.

[44] K. Allix, T.F. Bissyandé, J. Klein, Y. Le Traon, Androzoo: Collecting millions of android apps for the research community, in: Mining Software Repositories, MSR, 2016 IEEE/ACM 13th Working Conference on 2016 May 14, IEEE, 2016, pp. 468–471.

[45] https://apkpure.com/.

[46] https://www.apkmirror.com/.

[47] https://github.com/androguard/androguard.

[48] M.B. Kursa, W.R. Rudnicki, Feature selection with the Boruta package, J. Stat. Softw. 36 (11) (2010) 1–3, Available at https://www.jstatsoft.org/article/view/v036i11.

[49] http://danielhomola.com/2015/05/08/borutapy-an-all-relevant-feature-selection-method/.

[50] https://medium.com/the-artificial-impostor/feature-importance-measures-for-tree-models-part-i-47f187c1a2c3.

[51] N. Cristianini, J. Shawe-Taylor, An Introduction to Support Vector Machines and Other Kernel-Based Learning Methods, Cambridge university press, 2000.

[52] A.P. Bradley, The use of the area under the ROC curve in the evaluation of machine learning algorithms, Pattern Recognit. 30 (7) (1997) 1145–1159.

[53] Tensorflow framework, 2019, available at https://www.tensorflow.org/. (Access Date: 01.08.2019).

[54] Relu and Softmax available at https://keras.io/activations/.

[55] Adam optimizer, 2019, available at https://www.tensorflow.org/api_docs/python/tf/train/AdamOptimizer. (Access Date: 01.08.2019).

[56] Sigmoid Cross Entropy available at https://keras.io/losses/.