Hybrid Intelligent Android Malware Detection Using Evolving Support Vector Machine Based on Genetic Algorithm and Particle Swarm Optimization

Waleed Ali

Information Technology Department, Faculty of Computing and Information Technology, King Abdulaziz University, Rabigh, Kingdom of Saudi Arabia

Summary
The Android platform has become the most common mobile platform of smart mobile devices that attracts many users, developers and vendors. Accordingly, millions of Android applications have been created to offer many functionalities and services to users. However, the fast growth rate of such applications has led to a huge increase in the development and spread of Android malware applications by cyber attackers and criminals. In order to overcome the difficulties faced by the conventional signature-based methods, this paper suggests hybrid intelligent Android malware detection approaches based on evolving support vector machine with evolutionary algorithms in order to enhance Android malware detection. In the proposed hybrid intelligent evolving approaches, the optimization problem in support vector machine is solved using a genetic algorithm (GA) and a particle swarm optimization (PSO), referred to as Droid-HESVMGA and Droid-HESVMPSO, in order to help in increasing the accuracy of the Android malware detection. The experimental results showed that the proposed Droid-HESVMGA and Droid-HESVMPSO approaches achieved the best detection results and substantially outperformed the most popular machine learning classifiers and other existing hybrid malware detection approaches.

Key words:
Android, Malware, Support vector machine, Genetic algorithm, Particle swarm optimization.

1. Introduction

Recently, smartphones and mobile devices have become the most commonly used devices for personal and business use. Recent reports and studies have reported that the number of mobile device users has been increasing rapidly and will reach 6.1 billion by 2020 [1, 2].

Over the past few years, the Android platform has become the most common mobile platform of smart mobile devices as it is free and open-source. In addition, it can be customized simply by users, developers and vendors. Accordingly, millions of Android applications have been developed to offer many functionalities and services to users. Recent reports have indicated that over 2.5 million Android apps are available on Google Play store, which is considered the largest apps store [3].

The fast growth rate of Android applications has led to a huge increase in the development and spread of Android malware applications by cyber attackers and criminals. Primarily, the malware applications can be developed by Android apps developers and then distributed in third-party Android markets since there are often no constrains for Android apps developers. Even at the official Google Play store, many new malware apps are periodically discovered and not all Android malware Apps can be accurately detected, especially in the early stages of publication in the Google Android Market [4, 5].

Many commercial Android malware detection tools and anti-virus programs have used traditional signature-based methods, which are based on fixed identifiers called signatures [6-8] to detect the Android malware apps. However, the conventional signature-based approaches cannot detect the recently developed malware apps, especially zero-day Android malware apps [6-8]. Hence, there is a need to develop more effective and adaptive solutions for Android malware detection.

In order to overcome limitations of the conventional Android malware detection approaches, numerous single popular machine learning algorithms [6, 7, 9-16] have been trained and then applied to detect the Android malware apps. The support vector machine (SVM) algorithm has been commonly used in literature for detecting malware apps, as it has many advantages over the other machine learning techniques. However, only the classical SVM has been employed in Android malware detection, although the classical SVM is still not good enough compared to the advanced machine learning classifiers. Furthermore, it can be quite long time-consuming in the learning phase, as it is based on an analytical approach or complex mathematical calculations [17, 18].

In addition to single classifiers, ensemble learning methods [14, 19] and fusion approaches [8] for multiple machine learning classifiers have also been exploited in order to enhance the detection accuracy of Android malware apps. The single classifiers and ensemble methods have achieved better detection performance compared to traditional signature-based methods. However, the question: “Which

are the most effective machine learning techniques to maximize the performance of Android malware detection" is still a popular research subject [8, 20-22].

Although many intelligent approaches based on machine learning have become frequently utilized to detect Android malware apps in recent years, not much work has focused on the hybrid intelligent Android malware detection approaches. Hybrid intelligent approaches can be utilized to produce promising solutions with a higher detection accuracy of the Android malware apps, since they benefit from the advantages of the integrated algorithms and overcome their individual drawbacks.

Over the past few years, evolutionary algorithms have been effectively implemented in many recent applications and fields, including optimization, feature selection, pattern recognition, classification, and clustering. The genetic algorithm (GA) and particle swarm optimization (PSO) are the most well-known evolutionary algorithms used in complicated optimization problems. The GA is the most common evolutionary algorithm, based on imitation of biological evolution in chromosomes. The PSO is a well-known optimization and search algorithm inspired by the social and cooperative behavior of birds flocking, which belongs to a family of evolutionary algorithms.

In this paper, hybrid intelligent evolving approaches based on support vector machine with evolutionary algorithms are proposed to enhance Android malware detection. The proposed methods integrate the attractive advantages of evolutionary algorithms and the good performance of the support vector machine in order to produce hybrid intelligent evolving Android malware detection approaches with outstanding performance. In these proposed hybrid intelligent evolving approaches, genetic algorithm (GA) and particle swarm optimization (PSO) are adopted to effectively solve the optimization problem of support vector machine, so-called Droid-HESVMGA and Droid-HESVMPSO, in order to help in increasing the accuracy of the Android malware detection.

The remainder of the paper is arranged as follows. Section 2 reviews some existing works on intelligent Android malware detection based on static malware analysis. Section 3 describes the basics of android malware Applications including Android Architecture, Android Malware, and Android Permissions. The genetic algorithm and particle swarm optimization are described in Section 4. Section 5 provides an explanation of the support vector machine. Section 6 presents a methodology of the proposed hybrid intelligent android malware detection based on evolving SVM with GA and PSO. The experimental results of the proposed hybrid evolving intelligent android malware detection are analyzed and discussed in Section 7. Finally, the conclusion of the proposed work is given in Section 8.

2. Related Work

In order to overcome the difficulties faced by the conventional signature-based methods, machine learning techniques have been employed to discriminate the new malware from benign apps [8, 19, 20, 23]. In Android malware detection, the static analysis-based machine learning approach and the dynamic analysis-based machine learning approach are two popular approaches used to detect the malware apps.

In the static analysis-based approach, the machine learning models are trained based on static features of Android apps, which are extracted without installing or running these Android apps, in order to detect the malware applications. On the other hand, the dynamic-based approach requires installing, running and then monitoring the dynamic behavior of the Android application to collect the dynamics features in order to train the machine learning models.

The static malware analysis-based malware detection methods are easier, faster, and less resource-intensive [8, 20-22] compared with the dynamic analysis-based methods, since they do not require the android applications to be installed or run. Therefore, much emphasis in this study is focused on intelligent malware detection methods based on static malware analysis.

In recent years, several existing intelligent methods using machine learning have been developed based on static features of Android apps in order to detect Android malware applications. Support vector machine (SVM) [10-12, 16], naive Bayes [11, 13], k-NN [7, 9, 13], neural network [11], decision tree [7, 12], random forest [7, 11, 12, 14, 16], regularized logistic regression [6], neuro-fuzzy [24], hybrid evolving neuro-fuzzy [20], deep belief networks [15, 23], ensemble learning method [14, 19], fusion approach [8] and other machine learning classifiers have been trained and constructed using static features for detecting Android malware applications. Some previous intelligent Android malware detection methods based on the static malware analysis are summarized in Table 1.

By examining the existing works in Table 1, it can be observed that permissions, intents, and API calls were extracted and then used to train some popular conventional machine learning classifiers. In this study, instead of the classical machine learning classifiers used in the literature, alternative hybrid intelligent approaches based on evolving support vector machine with a genetic algorithm and a particle swarm optimization are suggested in order to enhance the performance of Android malware detection.

Table 1: Summary of the existing intelligent Android malware detection based on the static malware analysis

Approach	Features	Machine learning	Feature selection	Dataset source
DroidMat([9]	Permissions, intents, and API calls	k-means and k-NN	None	Google Play and Contagio Mobile
DREBIN [10]	Permissions, API calls and network addresses	SVM	None	Several markets, Google Play and Genome
Mining API calls and permissions for Android malware detection [13]	Permissions and API calls	Naive Bayes and k-NN	Correlation-based feature selection and information gain	Official, third party Android markets and Android malware Genome project
Static detection of Android malware by using permissions and API calls [11]	Permissions and API calls	Naive Bayes, SVM, MLP, random forest, and J48	Information gain	Anzhi Market and Contagio Mobile
Exploring Permission-induced Risk in Android Applications for Malicious Application Detection [12]	Individual permission and group of collaborative permissions.	SVM, decision trees, and random forest.	Forward selection (SFS) and principal component analysis (PCA)	Google Play , Mal Zhou and VirusShare
A probabilistic discriminative model for Android malware detection with decompiled source code [6]	API calls and permissions	Regularized logistic regression	Information gain and Chi-square	Google Play and Genome
K-ANFIS [24]	Permission-based features	kEFCM-based Adaptive Neuro-Fuzzy Inference System	Information gain ratio	Google Play and Genome
High Accuracy Android Malware Detection Using Ensemble Learning [14]	Permissions and API call features	Random forest as an ensemble learning method	None	McAfee's internal repository
DroidDetector [15]	Static analysis-based features and dynamic analysis -based features	Deep belief networks	Frequency analysis - based feature evaluation	Google Play, Contagio Community and Genome Project
EHNFC [20]	Permission-based features	Hybrid neuro-fuzzy classifier with evolving clustering	Information gain ratio	Google Play and Genome Project
Identification of malicious Android app using manifest and opcode features [16]	Static features from the manifest and executable files	SVM, random forest, and rotation forests	Entropy based Category Coverage Difference and Weighted Mutual Information	Google Play store and Drebin dataset
DAPASA [7]	Utilizing sensitive subgraphs to construct five features depicting invocation patterns.	Random forest, decision tree, k-NN, and PART	None	Google Play, Anzhi Market, Android Malware Genome Project and piggybacked families
Detecting Android malicious apps and categorizing benign apps with ensemble of classifiers [19]	11 types of static features from each app to characterize the behaviors of the app	Ensemble of multiple classifiers	SVM was used to sort the weight of each feature	Markets in China called Anzhi and Wild
DroidFusion [8]	Permissions, API calls and intents	Fusion approach	Information gain	DREBIN and Malgenome project

Compared to the existing studies, the hybrid evolving support vector machine proposed for Android malware detection is more accurate and effective than the usual support vector machine and other classical machine learning classifiers used in the literature.

As in most of the existing intelligent approaches, the most important permission features were extracted and then used to train the proposed hybrid evolving support vector machine. Compared to intents and API call features, the permission features are the most common and considered the first line of defense in the Android system. Furthermore, extracting the permission features and then applying them on machine learning models are easier, and consume fewer resources and require a shorter time [20, 24]. Thus, the permission features are more suitable to be used in the mobile environment to train the machine learning models.

3. Android Malware Applications

3.1 Android Architecture

The Android platform was introduced by Google on September 23, 2008 based on a Linux kernel, and has become a leading mobile operating system. Android consists of four layers: the Linux kernel layer, a native library layer, an application framework layer, and the application layer [25, 26].

The most significant layer that represents the core of the Android system is the Linux kernel layer. This layer is responsible for managing the services and hardware's functions. The native library layer deploys system libraries and the Android Dalvik virtual machine, which provides a

variety of functionalities and runtime environment for Android applications. The application framework layer is responsible for the Android APIs required to interact with the running apps and manage the essential functions. Eventually, all Android functionalities and applications are running and provided to the end-user by the application layer.

3.2 Android Malware

Malware is a malicious software and refers to several forms of hostile or intrusive applications, which are intentionally developed to damage, disrupt, steal, or primarily inflict some illegitimate and harmful actions on the computer or network. There are several common several types of Android malware, depending on their purpose, such as worm, virus, trojan, adware, spyware, rootkit, backdoor, keylogger, ransomware and remote administration tools (RAT).

Basically, Android applications are Java codes compiled by the Android SDK tools into an Android package (APK). The APK file is a type of archive files with a .apk suffix, which consists of the files and folders shown in Table 2 that are used by Android to install the application [27, 28].

Table 2: Components of Android package (APK)

Component	Description
AndroidManifest.xml	A meta-data XML file including information related to application's descriptions, package information and security permissions.
Classes.dex	A file that contains the source code of an Android application written in Java programming language compiled into .dex format (Dalvik Executable)
Resources.arsc	A binary XML file that contains precompiled application resources
Resources folder (res/)	A folder that includes non-pre-compiled resources that the application needs in runtime, such as pictures, layout, use of a database and data stored in the database, etc.
Assets (assets/)	An optional folder that contains application assets that can be retrieved by AssetManager
Libraries (lib/)	An optional folder that contains compiled code that is specific for different processors, such as arm, mips, x86, etc
META-INF	A folder that contains MANIFEST.MF file, APK signature, etc.

According to the McAfee threat report in September 2018 [29] there were over 25 million mobile malware applications as shown in Fig. 1. According to this report, over 2 million new mobile malware applications had been detected during the second quarter of 2018 only, mostly targeting Android, due to the vast distribution of Android devices, as well as the relatively open system for the distribution of apps.

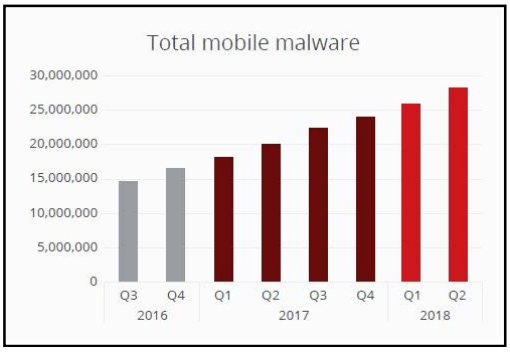

Fig. 1 The total number of mobile malware applications, mostly targeting Android [29]

Over the past few years, the fast growth rate of Android applications has led to a huge increase in developing and spreading Android malware applications by cyber attackers and criminals. Although numerous malware apps are frequently detected, other new Android malware Apps cannot be accurately discovered in the early stages by third-party Android markets, even by the official Market [4, 5].

Android malware apps can use different techniques in order to hijack the mobile device and access personal data. Primarily, repackaging, update attack and drive-by-download techniques are used to trick users into downloading the Android malware apps [27, 30]. In repackaging, some popular applications from legitimate sources are downloaded and disassembled by the malware developer. The malware developer can enclose the malware payload to these popular applications, and then resubmit them to official or other Android markets. Over 80% of Android malware is a repackaged application. In an update attack, the malware developer attaches only an update component to these popular applications, instead of attaching the whole payload into the application code. The update component can then download the entire malware payload in the app's runtime. The drive-by-download is a conventional social-engineering method implemented in the mobile devices field, through which the malware developer deceives the mobile user into installing interesting apps, which will perform other expected actions.

3.3 Android Permissions

In the Android platform, the security issue is mainly based on permissions-based mechanism, which is utilized to protect Android users from undesirable activities of some Android apps, such as accessing sensitive user data, system resources, and other app's data. The Android platform has more than 130 official permissions [30, 31, 32]. Some permissions are commonly requested by the malware and benign apps while other permissions are rarely requested.

At the installation time, some permissions are immediately granted by Android without user confirmation, while the

user's approval is required with other permissions, according to the category of the permission requested, either normal or potentially dangerous permission.

The low-risk permissions are classified under normal permission category, which are not particularly harmful and do not present any risk to the user's privacy or the device's operation such as INTERNET, ACCESS_NETWORK_STATE, and MODIFY_AUDIO_SETTINGS [20, 30]. On the other hand, the higher risk permissions could potentially affect the user's privacy, hardware, software or system. These high-risk permissions are categorized under the dangerous permission category. The malware apps are highly interested in requesting the dangerous permissions to gain the required privileges in order to access sensitive information. For example, READ_CONTACTS, WRITE_CONTACTS, CALL_PHONE, and SEND_SMS are four dangerous permissions, requiring the explicit approval of the user at the installation time [20, 30].

The permission request can be approved or rejected by the user without stopping the application, which will run with limited capabilities. In Android 6.0 or higher, the dangerous permissions must be granted by user at runtime, in the case where the user is not notified of any app permissions at the installation time. Even if the dangerous permissions are granted by the user at the installation time, the user can enable and disable permissions one-by-one in system settings at runtime [33].

4. Support Vector Machine

The support vector machine (SVM) was introduced by Vapnik [34] and has become one of the most popular machine learning techniques. SVM has many advantages over others. The generalization ability of SVM can be maximized, since SVM is trained to maximize the margin. In addition, there is a global optimum solution in SVM training. Furthermore, SVM is robust to outliers, because the margin parameter C controls the misclassification error. Therefore, SVM has been successfully applied in many complex classification applications.

Consider a set of training data vectors $X = \{x_1, \dots, x_n\}$, $x_i \in R^d$, and a set of corresponding labels $Y = \{y_1, \dots, y_n\}$, $y_i \in \{1, -1\}$. SVM aims to maximize the margin between the separating hyperplane and the closest instance in each class in order to obtain the ideal hyperplane between the two different classes. The hyperplane can be expressed as in Eq. (1).

$$(w.x) + b = 0, w \in R^d, b \in R \qquad (1)$$

where the vector w defines the boundary, x is the input vector of dimension d, and b is a scalar threshold.

The optimal hyperplane can be obtained as a solution to the following optimization problem.

$$\text{minimize } \frac{1}{2}\|w\|^2 \qquad (2)$$

$$\text{subject to } y_i\big((w.x_i) + b\big) \geq 1 \ , \forall i \qquad (3)$$

In real-world applications, the data are usually influenced by outliers, which are affected by noise. The decision boundaries can be softened by introducing a slack positive variable ξ for each training pattern. Eq. (4) is called the primal optimization of SVM.

$$\text{minimize } \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{L}\xi_i \qquad (4)$$

$$\text{subject to } y_i\big((w.x_i) + b\big) \geq 1 - \xi_i \ , \forall i \qquad (5)$$

where C is a positive regularization constant, which controls the degree of penalization of ξ. Therefore, C controls allowable errors in the trained solution: high C permits few errors while low C allows a higher proportion of errors in the solution.

To solve the convex optimization problem, Lagrangian multipliers α_i are used to produce the to the dual optimization problem, as shown in Eq. (6), which must be solved in order to find a separating maximum margin hyperplane for a given set of data points.

$$\text{maximize } \sum_{i=1}^{n}\alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} y_i y_j \, \alpha_i \alpha_j (x_i . x_j) \qquad (6)$$

$$\text{subject to } 0 \leq \alpha_i \leq C \text{ for all } i = 1, \dots, n \qquad (7)$$

$$\text{and } \sum_{i=1}^{n}\alpha_i y_i = 0 \qquad (8)$$

In most cases, the data points are not linearly separable. Thus, the SVM will transform the data to a higher-dimensional space and then classify them using the same principle as the linear case. A kernel function $K(x_i, x_j)$ is used to perform this transformation and the dot product in a single step. Thus, the final dual optimization problem using kernel function can be expressed using Eq. (9) to find a separating maximum margin hyperplane for non-separable data points.

$$\text{maximize } \sum_{i=1}^{n}\alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} y_i y_j \, \alpha_i \alpha_j K(x_i, x_j) \qquad (9)$$

$$\text{subject to } 0 \leq \alpha_i \leq C \text{ for all } i = 1, \dots, n \qquad (10)$$

$$\text{and } \sum_{i=1}^{n} \alpha_i y_i = 0 \qquad (11)$$

The dual optimization problem of SVM is usually solved by classical optimization methods such as Sequential Minimal Optimization (SMO) [35], Kernel Adatron (KA) [36, 37] and Quadratic Program (QP)[38]. However, these classical optimization methods are based on an analytical approach or complex mathematical calculations. Furthermore, and their performances are modest compared to those of the evolutionary algorithms used in this paper.

5. Evolutionary Algorithms

In the past few years, evolutionary algorithms have become a very popular research topic, which have been effectively employed in many applications and fields such as optimization, feature selection, pattern recognition, classification, and clustering. Evolutionary algorithms are a set of modern metaheuristic optimization algorithms based on the evolution of populations, which are primarily developed to solve complicated optimization problems. The most well-known evolutionary algorithms used in optimization problems are the genetic algorithm (GA) [39] and particle swarm optimization (PSO) [40].

5.1 Genetic Algorithm

The genetic algorithm [39] is the most common evolutionary algorithm based on the simulation of the biological evolution process in chromosomes. In other words, the genetic algorithm mimics the survival of the fittest among chromosomes of consecutive generations in order to solve a certain optimization problem. The genetic algorithm (GA) is commonly utilized to solve the optimization problems with a large search space [41, 42].
In GA, all possible candidate solutions construct the search space or population of a specific optimization problem. A basic GA mainly implements the following four major steps:

i. Encoding of chromosomes: Each candidate solution represents chromosome in a population, which is encoded with several genes. Each gene is a small part of a candidate solution, which can represent one parameter to be optimized.
ii. Initialization of the population: An initial population of chromosomes is randomly generated, which consists of the initial solutions of a specific optimization problem.
iii. Fitness evaluation: The GA computes the fitness value of each individual chromosome, which indicates the goodness of the solution provided by the individual chromosome. The chromosomes in the population are then evaluated using the fitness function. The fittest chromosomes will be given more opportunities to reproduce and evolve.
iv. Reproduction: As in biological evolution, a GA can recombine the fittest chromosomes to create new better chromosomes and solutions. The reproduction process is conducted through three genetic operators: selection, crossover, and mutation.
    - Selection: The better chromosomes are selected based on the fitness values to become parents to produce new chromosomes (offspring).
    - Crossover: In the crossover operator, GA randomly chooses a crossover point, where two parent chromosomes break, and then exchanges the chromosome parts after that point in order to create new offspring.
    - Mutation: The mutation operator changes the gene value in some randomly chosen location of the chromosome.

Some selected chromosomes are iteratively evolved to produce a new generation of new better solutions. The reproduction and fitness evaluation are repeated until the termination criterion is satisfied.

5.2 Particle Swarm Optimization

The particle swarm optimization algorithm (PSO) is a common population-based optimization algorithm tied to evolutionary computation, which was introduced by Kennedy and Eberhart [40]. PSO is a simpler and faster evolutionary algorithm and has fewer parameters compared to GA. Therefore, PSO has been widely applied in many problems and areas such as optimization, feature selection, pattern recognition, classification and clustering [42-46].
Unlike the chromosome's evolution in GA, PSO is inspired by the social behavior of birds flocking in interacting and cooperating to find food. Like evolutionary algorithms, a PSO population (called a swarm) consists of candidate solutions or individuals (called particles) which are randomly initialized. Each particle then moves in the search space with a velocity v in order to find the optimal solution. The particles learn over time based on their own experience and the experience of the other particles in the swarm.
Le $x_i = (x_{i1}, x_{i2}, x_{i3}, \dots, x_{iD})$ be the current position of particle i, and $v_i = (v_{i1}, v_{i2}, v_{i3}, \dots, v_{iD})$be the velocity of particle i, where D is the dimensionality of the search space. To find the best solution in PSO, each particle changes its velocity, as shown in Eq. (12) and Eq. (13), according to pbest and gbest, which represent the best previous position of a particle (personal best position) and the best position obtained by the whole population (global best position), respectively.

$$x_{id}^{t+1} = x_{id}^{t} + v_{id}^{t+1} \quad (12)$$

$$v_{id}^{t+1} = w * v_{id}^{t} + c_1 * r_1 * (p_{id} - x_{id}^{t}) + c_2 * r_2 (p_{gd} - x_{id}^{t}) \quad (13)$$

where d=1,2,3 …D, t represents the tth iteration, p_{id} and p_{gd} denote the pbest and gbest, w is inertia weight, c_1 and c_2 are acceleration parameters which are commonly set to 2.0, and r_1 and r_2 are random values in the range [0, 1].

6. Methodology

This section will describe the methodology of the proposed hybrid intelligent Android malware detection approaches using evolving support vector machine (SVM) based on genetic algorithm (GA) and particle swarm optimization (PSO). In the proposed approaches, GA and PSO are adopted to solve the dual optimization problem in SVM, so-called Droid-HESVMGA and Droid-HESVMPSO, in order to help in increasing the accuracy of the Android malware detection. Fig. 2 shows the methodology of the hybrid evolving support vector machine approach suggested for Android malware detection.

6.1 Collection of Malware and Benign Apps

Numerous researchers have collected and analyzed many malware and benign apps from several sources such as Google Play store [47], Genome [48], Contagiodump [49], VirusTotal [50], MalShare[51], VirusShare[52] , and theZoo [53].

In this study, we used the same dataset as that used in [20], which consists of 500 malware and benign apps, in order to evaluate the performance of the proposed methods. In the dataset used in this study, 250 benign apps and 250 malware apps were collected from official Google Play store [54] and Genome [55], respectively, which are the most common sources of benign and malware apps. These apps have many permission features that can be used as input features to train and test the proposed hybrid intelligent Android malware detection.

6.2 Feature Extraction

The extraction and selection of the important features of Android apps play an extremely important role in recognizing malware from benign apps. In this step, some popular permission features of apps are extracted from these Android apps since the permissions features are the most significant features that can be utilized in Android malware detection.

In the development phase of an Android app, the developers of Android apps must declare all the permissions required to access system resources using <uses-permission> tags in the AndroidManifest.xml [31] as shown in Fig. 3.

As can be seen from Fig. 3, AndroidManifest.xml is an XML file that includes the permission features of Android applications. In this study, Apktool was used to decompress the Android application package (APK file) and then extract the AndroidManifest.xml in order to obtain the permission features. Only 50 frequently requested permissions were collected from Android applications to be used as input features to train and test the proposed hybrid Droid-HESVMGA and Droid-HESVMPSO.

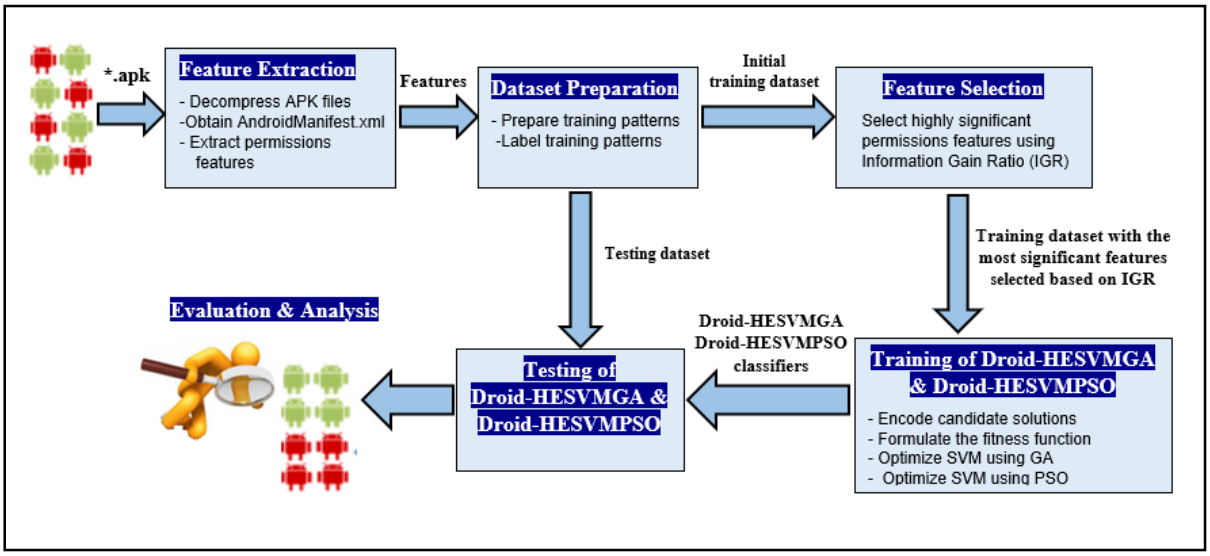

Fig. 2 A methodology of the proposed hybrid intelligent android malware detection using evolving SVM based on GA and PSO

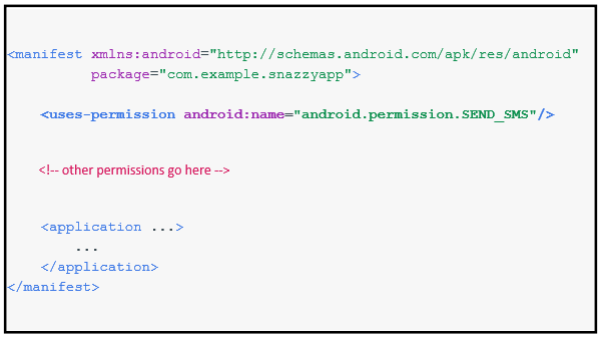

Fig. 3 The permissions declared to access system resources in AndroidManifest.xml of Android apps

6.3 Dataset Preparation

In this phase, the permission features of Android apps are converted into numerical form to be effectively used to train and construct the proposed hybrid Droid-HESVMGA and Droid-HESVMPSO. Initially, a base vector is prepared, which includes a set of the frequently requested permissions that can be effectively utilized in Android malware detection. For each Android app, a binary vector of permission features is then created as an instance of the base vector.

Each Android app represents a single training pattern, which is encoded with a binary vector of permission features and a class label indicates whether the Android app is benign or malware. A permission feature is assigned to binary value based on whether the permission feature is requested or not by the Android apps. The permission feature is represented by 1 in the binary vector if it is requested by the app. Otherwise, the permission feature is set to 0 in the binary vector. Furthermore, the last value in the binary vector represents the class of the Android app, either a malware or a benign app.

6.4 Feature Selection

In the mobile environment, feature selection is a vital step used to remove redundant and irrelevant permission features that can produce noisy data, causing a negative impact on the performance of intelligent classifiers. Therefore, a feature selection method should be used to identify the most significant permissions that can be effective in distinguishing malware from benign apps.

Generally, there are two primary feature selection approaches used in data mining: the filter approach and the wrapper approach. The methods under the filter feature selection approach are easier and faster compared to the methods of the wrapper approach, since they analyze and evaluate the features without training of the classifiers. Hence, the feature selection methods under the filter approach are more suitable to be used in the mobile environment.

In this study, one of the most common filter-based methods, known as information gain ratio, is applied to select highly significant permission features of Android apps.

Eq. (14) computes the information gain, Gain(S, A) of a feature A, relative to a collection of examples S.

$$Gain(S,A) = Entropy(S) - \sum_{v \in Value(A)} \frac{|S_v|}{|S|} Entropy(S_v) \quad (14)$$

where Value(A) is the set of all possible values for a feature A, and S_v is the subset of S for which feature A has value v. Entropy(S) is defined as Eq. (15):

$$Entropy(S) = -\sum_{j=1}^{/C/} Pr(c_j) \log_2 Pr(c_j) \quad (15)$$

where $Pr(c_j)$ denotes the probability of class c_j in S. It is the number of examples of class c_j in S divided by the total number of examples in S.

The information gain ratio is an enhancement of the information gain that decreases its bias toward high-branch attributes. The information gain ratio is employed in feature selection to achieve better performance. In the information gain ratio, Eq. (16) is used in order to evaluate the features:

$$Gain\ ratio(S,A) = \frac{Gain(S,A)}{Split\ information(S,A)} \quad (16)$$

where Split information (S, A) is computed using Eq. (17):

$$Split\ information(S,A) = -\sum_{i=1}^{k} \frac{/S_i/}{/S/} \log_2 \frac{/S_i/}{/S/} \quad (17)$$

where S_1 through S_k are the k subsets of examples resulting from partitioning S by the k-values feature A.

In the proposed hybrid Droid-HESVMGA and Droid-HESVMPSO, only the best 25 permission features that have a high impact on Android malware detection are selected using the information gain ratio (IGR), in order to contribute toward enhancing the performance of the evolving support vector machine classifiers suggested to detect the Android malware apps.

6.5 Training of Droid-HESVMGA and Droid-HESVMPSO

In this phase, the proposed Droid-HESVMGA and Droid-HESVMPSO are trained using the prepared training dataset with the Android permission features selected by information gain ratio. The significant permission features are used as input features of Droid-HESVMGA and Droid-HESVMPSO, which are trained in order to classify the Android apps into two classes, either the malware or benign apps.

Unlike the conventional SVM, the GA and PSO are used in the proposed Droid-HESVMGA and Droid-HESVMPSO to solve the dual optimization problem in support vector machine in order to increase accuracy of the Android malware detection.

In the proposed Droid-HESVMGA and Droid-HESVMPSO, each candidate solution or individual is represented by a vector and denoted as a chromosome in population GA or a particle in the PSO swarm. The GA chromosome and position of each PSO particle is represented by a vector of real values, in which each value represents the value of the Lagrange multiplier for a training example as shown in Fig. 4. Fig. 4 illustrates an example of encoding the Lagrange multipliers vector in a GA chromosome and PSO particle in the proposed Droid-HESVMGA and Droid-HESVMPSO.

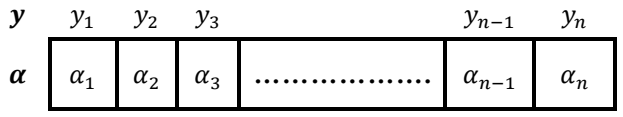

Fig. 4 Encoding of Lagrange multipliers vector in GA chromosome and PSO particle

In the evolutionary algorithms, once the candidate solutions are encoded, the fitness function is used to evaluate the candidate solutions or individuals. In order to find a separating maximum margin hyperplane of SVM for a given set of data points, Eq. (9) is used as the fitness function in the proposed Droid-HESVMGA and Droid-HESVMPSO to evaluate the GA chromosomes and PSO positions.

In the proposed Droid-HESVMGA, an initial population of chromosomes is randomly generated, which represent the values of Lagrange multipliers for training patterns. The chromosomes' performances are then computed and evaluated by the fitness function shown in Eq. (9). The GA will stop the search and return the optimal vector of Lagrange multipliers if the good fitness or maximum generations number is reached. Otherwise, the GA implements selection, crossover, and mutation to produce a new generation of chromosomes in order to find the optimal vector of Lagrange multipliers that can maximize the performance of SVM. The fittest chromosomes are the most appropriate candidate for mating to produce a new generation. Crossover and mutation are then employed to produce child chromosomes, used as alternative chromosomes to their parent chromosomes in the GA population. The parent chromosomes are then chosen to exchange the chromosome genes using the crossover process to offer a child chromosome with genetic materials. In GA mutation, a gene in the child chromosome can be changed to a random value between 0 and C in the proposed Droid-HESVMGA.

In the proposed Droid-HESVMPSO, an initial swarm of particles is randomly generated; each particle represents the value of the Lagrange multiplier for a certain training example. Each particle's fitness is then computed using Eq. (9) and evaluated accordingly. The PSO fitness function aims at finding a separating maximum margin hyperplane for given training examples. If the current particle fitness is better than the best fitness of that particle (pbest), then the new pbest will be updated to the current particle fitness. The global best fitness(gbest) is then updated to the particle with the best fitness value of all the particles. If the stopping criteria (sufficiently good fitness or maximum iterations) are met, the PSO will terminate the search and return the optimal values of the Lagrange multipliers. Otherwise, the pbest and gbest are utilized to update the velocity and position for every particle using Eq. (12) and Eq. (13). This process is repeated until the stop conditions are met.

After solving the optimization problem and obtaining the Lagrange multipliers by using the GA and PSO, the proposed Droid-HESVMGA and Droid-HESVMPSO can be used in Android malware detection. The proposed Droid-HESVMGA and Droid-HESVMPSO use the decision Eq. (18) to classify each input vector x into positive or negative class. In Android malware detection, the positive class refers to the malware apps, while the negative class represents the benign apps.

$$y(x) = sgn\left(\sum_{i=1}^{n} \alpha_j y_j K(x_i, x) + b\right) \qquad (18)$$

In the proposed Droid-HESVMGA and Droid-HESVMPSO, the radial basis function (RBF) defined as Eq. (19) was used as the kernel function $K(x_i, x)$, since it achieved a better performance in many applications compared to other kernel functions. The parameter γ represents the width of the RBF.

$$K(x_i, x) = exp(-\gamma \|x_i - x\|^2), \gamma > 0 \qquad (19)$$

7. Analysis and Discussion of Results

7.1 Dataset Collection and Preparation

In this study, the dataset with 500 Android apps used by [20] was adopted in our experiments in order to train and evaluate the proposed hybrid intelligent android malware detection based on the evolving support vector machine: Droid-HESVMGA and Droid-HESVMPSO. In this dataset, 250 malware apps were collected from official Google Play [54] while 250 malware apps were collected from Genome [55], which is commonly used in the literature to collect malware apps.

In order to prepare the training dataset, the permission features of these Android apps were extracted and

converted to binary forms based on whether the permission feature is requested or not by the Android apps. Accordingly, the best 25 permissions features were selected by using information gain ratio in order to help in improving the performance of the proposed hybrid intelligent Android malware detection method based on evolving support vector machine.

7.2 Evaluation Measures

In order to evaluate the proposed methods, Droid-HESVMGA and Droid-HESVMPSO were trained and evaluated using 5-fold cross-validation.

In our experiments, we used five popular metrics, which are commonly used in the literature for detecting malware apps, to evaluate the performance of the proposed Droid-HESVMGA and Droid-HESVMPSO. Correct classification rate, true positive rate, false positive rate, false negative rate and area under ROC curve were calculated in order to judge the effectiveness of the proposed Droid-HESVMGA and Droid-HESVMPSO. The correct classification rate (CCR) is the rate of malware and benign apps that are correctly classified with respect to all Android apps. True positive rate (TPR) is the rate of malware apps classified as malware out of total malware apps. False positive rate (FPR) is the rate of benign apps classified as malware out of total benign apps. False negative rate (FNR) is the rate of malware apps classified as benign out of total malware apps. The area under ROC curve (AUC) is a measure used to evaluate the trade-off between TPR and FPR.

Table 3 shows the measures used to evaluate the performance of proposed evolving support vector machine classifiers in Android malware detection. In Table 3, true positive (TP) is the number of correctly classified malware apps, false negative (FN) is the number of incorrectly classified malware apps, true negative (TN) is the number of correctly classified benign apps, and false positive (FP) is the number of incorrectly classified benign apps.

7.3 Comparison Against Popular Machine Learning Classifiers

In this study, the proposed Droid-HESVMGA and Droid-HESVMPSO were trained and compared with two common implementations of SVM, known as LibSVM [56] and mySVM [57], which use the classical optimization techniques to solve the quadratic programming problem.

In all SVMs, RBF was used as the kernel function while the best parameters C (margin softness) and γ (RBF width) were obtained by using a grid search algorithm in order to achieve the best performance for Android malware detection. In addition, the proposed Droid-HESVMGA and Droid-HESVMPSO were compared with other four machine learning classifiers commonly used in the literature to detect the Android malware applications: back-propagation neural network (BPNN), naïve Bayes classifier (NB), random forest (RF), and k-Nearest neighbour (kNN). In the proposed Droid-HESVMGA and Droid-HESVMPSO, it was found by a trial-and-error basis that the parameters settings of the GA and PSO shown in Tables 4 and 5 produced good results.

Table 3: The performance measures used to evaluate the proposed methods

Measure name	Formula (%)	Description
Correct classification rate (CCR)	$CCR = \frac{TP + TN}{TP + FP + FN + TN} \times 100$	The rate of malware and benign apps correctly classified with respect to all the apps.
True positive rate (TPR)	$TPR = \frac{TP}{TP + FN} \times 100$	Rate of malware apps classified as malware out of total malware apps.
False positive rate (FPR)	$FPR = \frac{FP}{FP + TN} \times 100$	Rate of benign apps classified as malware out of total benign apps.
False negative rate (FNR)	$FNR = \frac{FN}{FN + TP} \times 100$	Rate of malware apps classified as benign out of total malware apps.
Area under ROC curve (AUC)	$AUC = \frac{1 + TPR - FPR}{2} \times 100$	This measures the trade-off between TPR and FP

Table 4: Parameters settings of GA used in the proposed Droid-HESVMGA

Parameter	Value
Population size	20
Maximum generation	1000
Crossover probability	0.9
Mutation type	Switching mutation
Selection scheme	Tournament (0.75)

Table 5: Parameters settings of PSO used in the proposed Droid-HESVMPSO

Parameter	Value
Number of particles	20
Maximum iterations (generations)	1000
C1	2
C2	2
Stop condition	maximum number of iterations

Table 6 shows the performance in terms of CCR, TPR, FPR, FNR, and AUC for the proposed Droid-HESVMGA and Droid-HESVMPSO against other machine learning

classifiers used in Android malware detection. It is clear from Table 6 that the proposed Droid-HESVMGA and Droid-HESVMPSO outperformed BPNN, NB, RF, kNN, mySVM, and LibSVM in most of the performance measures.

Table 6: Comparison of the proposed Droid-HESVMGA and Droid-HESVMPSO against popular machine learning classifiers used in Android malware detection

	CCR	TPR	FPR	FNR	AUC
BPNN	87.80	91.20	15.60	8.80	95.40
NB	74.80	99.20	49.60	0.80	70.60
RF	88.80	89.60	12.00	10.40	97.30
kNN	86.80	77.60	4.00	22.40	95.70
mySVM	82.00	85.60	21.60	14.40	96.10
LibSVM	88.20	91.60	15.20	8.40	96.20
Proposed Droid-HESVMGA	95.60	98.00	6.80	2.00	96.90
Proposed Droid-HESVMPSO	94.80	98.00	8.40	2.00	96.00

As can be observed from Table 6, the proposed Droid-HESVMGA and Droid-HESVMPSO achieved much better CCR than other machine learning classifiers used in Android malware detection. In particular, the proposed Droid-HESVMGA produced the highest CCR (95.60%), followed by Droid-HESVMPSO (94.80%), among the other machine learning classifiers. This indicates that the proposed Droid-HESVMGA and Droid-HESVMPSO were able to correctly detect both malware and benign apps with respect to all the Android apps.

In terms of other measures, the results shown in Table 6 demonstrate that the proposed Droid-HESVMGA and Droid-HESVMPSO also achieved better performance in both TPR and FPR compared to other machine learning classifiers used in Android malware detection. Actually, there is a trade-off between TRR and FPR. Therefore, a balanced performance between TPR and FPR should be provided in a good malware detection system.

Although the highest TPR was accomplished by NB, NB also produced the poorest FPR among other machine learning classifiers used in this study. This was due to NB produced unbalanced detection between malware and benign apps. This negatively affected the overall accuracy and AUC of NB used in Android malware detection. On the other hand, the proposed Droid-HESVMGA and Droid-HESVMPSO produced balanced detection performance between the positive and negative classes in both TRR and FPR. This indicates that the proposed Droid-HESVMGA and Droid-HESVMPSO were able to precisely detect both malware and benign apps. Consequently, the proposed Droid-HESVMGA and Droid-HESVMPSO achieved better performance in terms of the overall accuracy (CCR), TPR , FPR and AUC compared to the other machine learning classifiers.

In addition, Table 6 presents the performance in terms of FNR for the proposed Droid-HESVMGA and Droid-HESVMPSO compared to other machine learning classifiers. FNR is also an important measure in Android malware detection, since it calculates the rate of malware apps classified as benign out of total malware apps. It can be seen in Table 6 that the proposed Droid-HESVMGA and Droid-HESVMPSO achieved lower FNR compared to most of the other machine learning classifiers used in Android malware detection. Only 2% of malware apps were incorrectly classified as benign apps by the proposed Droid-HESVMGA and Droid-HESVMPSO.

7.4 Comparison Against Other Hybrid Android Malware Detection Works

In this section, the proposed Droid-HESVMGA and Droid-HESVMPSO were compared with other existing hybrid malware detection approaches, which combined several algorithms into classifiers to enhance the performance of malware detection. The proposed Droid-HESVMGA and Droid-HESVMPSO were compared to other previous works: evolving hybrid neuro-fuzzy classifier (EHNFC) [20], dynamic evolving fuzzy inference system (DENFIS) [20, 58] and adaptive fuzzy inference system with triangular membership function (TRIMF–ANFIS) [20]. For a fair comparison, the proposed Droid-HESVMGA and Droid-HESVMPSO were trained and then evaluated using the same dataset used in these previous works.

The results in Table 7 clearly depict the overall classification accuracy (CCR), TPR, FPR, FNR and AUC for the proposed Droid-HESVMGA and Droid-HESVMPSO compared to those of EHNFC, DENFIS, and TRIMF–ANFIS.

Table 7: Comparison of the proposed Droid-HESVMGA and Droid-HESVMPSO against other hybrid Android malware detection works

	CCR	TPR	FPR	FNR	AUC
EHNFC	90.00	88.24	5.00	5.00	95.00
DENFIS	82.20	87.50	19.05	12.50	92.20
TRIMF–ANFIS	88.00	78.95	11.11	21.05	93.00
Proposed Droid-HESVMGA	95.60	98.00	6.80	2.00	96.90
Proposed Droid-HESVMPSO	94.80	98.00	8.40	2.00	96.00

In terms of CCR, the results in Table 7 show that the proposed Droid-HESVMGA accomplished the highest accuracy (95.60%), followed by the proposed Droid-HESVMPSO (94.80%), EHNFC (90.00%), TRIMF–ANFIS (88.00%), and EHNFC (82.20%).

In terms of TPR, FPR, and FNR, the proposed Droid-HESVMGA and Droid-HESVMPSO achieved much better TPR than EHNFC, DENFIS, and TRIMF–ANFIS. Furthermore, the lowest FNR (only 2.00%) was accomplished by the proposed Droid-HESVMGA and Droid-HESVMPSO. Meanwhile, the proposed Droid-HESVMGA and Droid-HESVMPSO produced lower FPR compared to the FPRs obtained by DENFIS and TRIMF–ANFIS. This was primarily due to the capability of the proposed Droid-HESVMGA and Droid-HESVMPSO to successfully detect both malware and benign apps. On the

other hand, EHNFC performed unbalanced performance between malware and benign apps, since it produced lower TPR and the best FPR among other malware detection approaches.

The best TPR and FPR of the proposed Droid-HESVMGA and Droid-HESVMPSO produced the best AUC (96.00%) among the AUCs achieved by other malware detection approaches. This was because the AUC metric is used to measure the trade-off between TPR and FPR, as shown in Table 3. The results in Table 7 demonstrate that the proposed Droid-HESVMGA had the highest AUC (96.90%), followed by Droid-HESVMPSO (96.00%), EHNFC (95.00%), TRIMF–ANFIS (93.00%), and then DENFIS (92.20%).

8. Conclusion and Future Work

This paper has presented a methodology for proposed hybrid intelligent Android malware detection approaches using evolving support vector machine based on evolutionary algorithms. In the proposed hybrid intelligent Android malware detection approaches, GA and PSO were exploited in SVM to solve the dual optimization problem, referred to as Droid-HESVMGA and Droid-HESVMPSO, in order to enhance the detection accuracy of Android malware apps. The proposed Droid-HESVMGA and Droid-HESVMPSO produced promising solutions with higher detection accuracy of the Android malware apps, since they had the potential gains derived from exploiting both GA and PSO optimization methods in SVM classifier. The experimental results demonstrated that the proposed Droid-HESVMGA and Droid-HESVMPSO accomplished much better CCRs than popular machine learning classifiers and other existing hybrid malware detection approaches used in Android malware detection. Furthermore, the best TPR, FPR, FNR and AUC measures were accomplished by the proposed Droid-HESVMGA, followed by the proposed Droid-HESVMPSO.

In this study, the proposed Droid-HESVMGA and Droid-HESVMPSO were trained using only the permission features of Android malware applications. The proposed Droid-HESVMGA and Droid-HESVMPSO can be improved further by utilizing the intents and API call features of Android malware applications in the training phase.

Acknowledgment

This project was funded by the Deanship of Scientific Research (DSR), King Abdulaziz University, Jeddah, under grant No. (D-212-830-1440). The authors, therefore, gratefully acknowledge the DSR technical and financial support.

References

[1] A. Boxall, In 2020 6.1 Billion People Will Use A Smartphone Digital Trends, 2019. On the WWW, URL https://www.digitaltrends.com/mobile/smartphone-users-number-6-1-billion-by-2020

[2] N. Milosevic, A. Dehghantanha, and K. K. R. Choo, "Machine learning aided Android malware classification," Computers and Electrical Engineering, vol. 61, pp. 266–274, 2017.

[3] AppBrain, Number of Android applications on the Google Play store | AppBrain, 2019. On the WWW, URL https://www.appbrain.com/stats/number-of-android-apps

[4] W. J. Buchanan, S. Chiale, and R. Macfarlane, "A methodology for the security evaluation within third-party Android Marketplaces," Digital Investigation, vol. 23, pp. 88–98, 2017.

[5] F. Martinelli, I. Matteucci, M. Petrocchi, A. Saracino, G. Dini, and D. Sgandurra, "Risk analysis of Android applications: A user-centric solution," Future Generation Computer Systems, vol. 80, pp. 505–518.

[6] L. Cen, C. S. Gates, L. Si, and N. Li, "A probabilistic discriminative model for Android malware detection with decompiled source code," IEEE Trans. Depend. Secure Comput., vol. 12(4), pp. 400–412.

[7] Fan Ming, Liu Jun, Wang Wei, Li Haifei, Tian Zhenzhou, and Liu Ting, "DAPASA: Detecting Android piggybacked apps through sensitive subgraph analysis," IEEE Trans. Inf. Forensics Security, vol. 12(8), pp. 1772–1785, 2017.

[8] S. Y. Yerima, and S.Sezer, "DroidFusion: A Novel Multilevel Classifier Fusion Approach for Android Malware Detection," IEEE Transactions on Cybernetics, vol. 49(2), pp. 453–466, 2019.

[9] D.-J. Wu, C.-H. Mao, T.-E. Wei, H.-M Lee, and K.-P Wu, "DroidMat: Android malware detection through manifest and API calls tracing," In Proc. 7th Asia Joint Conf. Inf. Security (Asia JCIS), pp. 62–69, 2012.

[10] Arp, Daniel, M. Spreitzenbarth, M. Hübner, H. Gascon, and K. Rieck, "Drebin: Effective and Explainable Detection of Android Malware in Your Pocket," In Proceedings 2014 Network and Distributed System Security Symposium, pp. 1–15, 2014.

[11] P. P. K. Chan, and W.-K. Song, "Static detection of Android malware by using permissions and API calls," In Proc. Int. Conf. Mach. Learn. Cybern., Lanzhou, pp. 82–87, 2014.

[12] Wang, Wei, X. Wang, D. Feng, J. Liu, Z. Han, and X. Zhang," Exploring Permission-Induced Risk in Android Applications for Malicious Application Detection," IEEE Transactions on Information Forensics and Security, vol. 9(11), pp. 1869 – 1882, 2014.

[13] A. Sharma and S. K. Dash, "Mining API calls and permissions for Android malware detection," In Cryptology and Network Security, Cham, Switzerland: Springer Int., pp. 191–205, 2014.

[14] Yerima , S. Sezer, and Igor Muttik, "High Accuracy Android Malware Detection Using Ensemble Learning," IET Information Security, vol. 9(6), pp. 313 – 320, 2015.

[15] Yuan, Zhenlong, Yongqiang Lu, and Yibo Xue., "Droiddetector: Android Malware Characterization and Detection Using Deep Learning," Tsinghua Science and Technology, vol. 21(1), pp. 114-123, 2016.

[16] M. V. Varsha, P. Vinod, and K. A Dhanya. "Identification of malicious Android app using manifest and opcode features," J. Comput. Virol. Hacking Tech., vol. 13(2), pp. 125–138, 2017.
[17] M. L. Dantas Dias, and A. R. R. Neto, "Evolutionary support vector machines: A dual approach," In 2016 IEEE Congress on Evolutionary Computation, pp. 2185–2192, 2016.
[18] I. Mierswa, "Evolutionary Learning with Kernels: A Generic Solution for Large Margin Problems," In Proceedings of the 8th annual conference on Genetic and evolutionary computation - GECCO '06 (p. 1553), 2006.
[19] W. Wang, Y. Li, X. Wang, J. Liu, and X. Zhang, "Detecting Android malicious apps and categorizing benign apps with ensemble of classifiers," Future Generation Computer Systems, vol. 78, pp. 987–994, 2018.
[20] A. Altaher, "An improved Android malware detection scheme based on an evolving hybrid neuro-fuzzy classifier (EHNFC) and permission-based features," Neural Computing and Applications, vol. 28(12), pp. 4147–4157, 2017.
[21] A. T. Kabakus, and I. A. Dogru, "An in-depth analysis of Android malware using hybrid techniques," Digital Investigation, vol. 24, pp. 25–33, 2018.
[22] C. Zhao, C. Wang, and W. Zheng, "Android Malware Detection Based on Sensitive Permissions and APIs," In International Conference on Security and Privacy in New Computing Environments (SPNCE), pp. 96–104, 2019.
[23] E. M. B. Karbab, M. Debbabi, A. Derhab, and D. Mouheb, "MalDozer: Automatic framework for android malware detection using deep learning," Digital Investigation, vol. 24, pp. S48–S59, 2018.
[24] A. Shubair, and A. Altaher, "Intelligent Approach for Android Malware Detection," KSII Transactions on Internet and Information Systems, vol. 9(8), pp. 2964 – 2983, 2015.
[25] K. Tam, A. Feizollah, N. B. Anuar, R. Salleh, and L. Cavallaro, "The Evolution of Android Malware and Android Analysis Techniques," ACM Computing Surveys, vol. 49(4), pp. 1–41, 2017.
[26] H. S. Ham, and M. J. Choi, "Analysis of Android malware detection performance using machine learning classifiers," In International Conference on ICT Convergence, 2013. https://doi.org/10.1109/ICTC.2013.6675404
[27] A. Guerra, "APPLICATION OF FULL MACHINE LEARNING WORKFLOW FOR MALWARE DETECTION IN ANDROID ON THE BASIS OF SYSTEM CALLS AND PERMISSIONS," MS Thesis, TALLINN UNIVERSITY OF TECHNOLOGY, School of Information Technologies, 2018. See also URL https://digi.lib.ttu.ee/i/?10770
[28] Yerima, S. Sezer, and G. McWilliams, "Analysis of Bayesian classification-based approaches for Android malware detection," IET Information Security, vol. 8(1), pp. 25-36, 2014.
[29] McAfee, McAfee Labs Threats Report, 2018. On the WWW, URL https://www.mcafee.com/enterprise/en-us/assets/reports/rp-quarterly-threats-sep-2018.pdf
[30] N. Peiravian, and X. Zhu, "Machine learning for Android malware detection using permission and API calls," In Proceedings - International Conference on Tools with Artificial Intelligence, ICTAI, pp. 300–305, 2013.
[31] Android, Manifest.permission | Android Developers, 2019. On the WWW, URL https://developer.android.com/reference/android/Manifest.permission#summary
[32] F. Idrees, M. Rajarajan, M. Conti, T. M. Chen, and Y. Rahulamathavan, "PIndroid: A novel Android malware detection system using ensemble learning methods," Computers and Security, vol. 68, pp. 36–46, 2017.
[33] Google, Control your app permissions on Android 6.0 and up - Google Play Help, 2019. On the WWW, URL https://support.google.com/googleplay/answer/6270602?hl=en
[34] V. Vapnik, "The nature of statistical learning theory," (2nd edition), New York: Springer, 1995.
[35] J. C. Platt, "Fast training of support vector machines using sequential minimal optimization," In Advances in Kernel Methods - Support Vector Learning. Cambridge, MA, USA: MIT Press, 1999.
[36] J. K. Anlauf and M. Biehl, "The adatron: An adaptive perceptron algorithm," Europhysics Letters, vol. 10(7), pp. 687, 1989.
[37] T-T. Frie, N. Cristianini, and C. Campbell, "The kernel-adatron algorithm: a fast and simple learning procedure for support vector machines," In Machine Learning: Proceedings of the Fifteenth International Conference (ICML'98), Citeseer, pp. 188–196, 1998
[38] P. E. Gill, W. Murray, and M. H. Wright, Practical optimization, 1981.
[39] D. E. Goldberg, "Genetic Algorithms in Search Optimization and Machine Learning," Addison-Wesley, 1989.
[40] J. Kennedy and R. Eberhart, "Particle swarm optimization," In IEEE International Conference on Neural Networks, pp. 1942–1948, 1995.
[41] B. Chakraborty, "Evolutionary Computational Approaches to Feature Subset Selection," International Journal of Soft computing and Bioinformatics, vol. 1(2), pp. 59-65, 2010.
[42] A. Kawamura, and B. Chakraborty, "A hybrid approach for optimal feature subset selection with evolutionary algorithms," Proceedings - 2017 IEEE 8th International Conference on Awareness Science and Technology, ICAST 2017, pp. 564–568, 2018.
[43] M-Y Cho, and T. T. Hoang, "Feature Selection and Parameters Optimization of SVM Using Particle Swarm Optimization for Fault Classification in Power Distribution Systems," Computational Intelligence and Neuroscience, Article ID 4135465, 9 pages, 2017.
[44] L. M. Abualigah, A. T. Khader, and E. S. Hanandeh, "A new feature selection method to improve the document clustering using particle swarm optimization algorithm," Journal of Computational Science, vol. 25, pp.456-466, 2018.
[45] J. Wei, Z. Jian-Qi, and Z. Xiang, "Face recognition method based on support vector machine and particle swarm optimization," Expert Systems with Applications, vol. 38(4): pp. 4390-4393, 2011.
[46] D. O'Neill, A. Lensen, B. Xue, and M. Zhang, "Particle Swarm Optimisation for Feature Selection and Weighting in High-Dimensional Clustering,". In 2018 IEEE Congress on Evolutionary Computation(CEC), 2018.
[47] Google Play, Google Play Store, 2019. On the WWW, URL https://play.google.com/store?hl=en

[48] Genome, Android Malware Genome Project, 2019. On the WWW, URL http://www.malgenomeproject.org
[49] Contagio, Contagio Mobile: mobile malware mini dump, 2019. On the WWW, URL http://contagiominidump.blogspot.co.uk
[50] VirusTotal, VirusTotal for Android, 2019. On the WWW, URL https://www.virustotal.com/en/documentation/mobile-applications
[51] MalShare, MalShare project, 2019. On the WWW, URL http://malshare.com/about.php
[52] VirusShare,VirusShare.com, 2019. On the WWW, URL https://virusshare.com
[53] theZoo, theZoo aka Malware DB, 2019. On the WWW, URL http://ytisf.github.io/theZoo
[54] Google Play, Google Play Store, 2014. On the WWW, URL https://play.google.com/store?hl=en
[55] Genome, Android Malware Genome Project, 2014. On the WWW, URL http://www.malgenomeproject.org
[56] C. C. Chang, and C. J. Lin. LIBSVM: A library for support vector machines, 2001. http://www.csie.ntu.edu.tw/~cjlin/libsvm
[57] S. Ruping, mySVM Manual, Universit¨at Dortmund, Lehrstuhl Informatik VIII, 2000. http://www-ai.cs.tu-dortmund.de/SOFTWARE/MYSVM/index.html
[58] N. Kasabov, Q. Song, "DENFIS: dynamic evolving neuralfuzzy inference system and its application for time-series prediction," IEEE Trans Fuzzy Syst, vol. 10(2), pp. 144–154, 2002.

Waleed Ali received his B.Sc. in Computer Science from Faculty of Science, Taiz University, Yemen in 2005. He obtained his M.Sc and Ph.D (Computer Science) from Faculty of Computing, Universiti Teknologi Malaysia(UTM), Malaysia in 2009 and 2012 respectively. Currently, he is Assistant Professor in IT department, Faculty of Computing and Information Technology in Rabigh, King Abdulaziz University since September 2013. He has published many papers in international journals, conferences and book chapters. His research interests include Intelligent Web caching, Intelligent Web prefetching, Web usage mining, Intelligent phishing website detection, Intelligent Android malware detection, and machine learning techniques and their applications.