Intelligent Approach for Android Malware Detection

Shubair Abdulla[1] and Altyeb Altaher[2]
College of Education, Instructional and Learning Technology Department,
Sultan Qaboos University, Muscat, Oman
[shubair@squ.edu.om]
Faculty of Computing and Information Technology, King Abdulaziz University,
Rabigh -21911,Saudi Arabia
[aaataha@kau.edu.sa]
*Corresponding author: Shubair Abdulla

Received February 16, 2015; revised May 8, 2015; accepted June 23, 2015; published August 31, 2015

Abstract

As the Android operating system has become a key target for malware authors, Android protection has become a thriving research area. Beside the proved importance of system permissions for malware analysis, there is a lot of overlapping in permissions between malware apps and goodware apps. The exploitation of them effectively in malware detection is still an open issue. In this paper, to investigate the feasibility of neuro-fuzzy techniques to Android protection based on system permissions, we introduce a self-adaptive neuro-fuzzy inference system to classify the Android apps into malware and goodware. According to the framework introduced, the most significant permissions that characterize optimally malware apps are identified using Information Gain Ratio method and encapsulated into patterns of features. The patterns of features data is used to train and test the system using stratified cross-validation methodologies. The experiments conducted conclude that the proposed classifier can be effective in Android protection. The results also underline that the neuro-fuzzy techniques are feasible to employ in the field.

Keywords: Android, neuro-fuzzy, malware detection, system permissions, classification

http://dx.doi.org/10.3837/tiis.2015.08.012 **ISSN : 1976-7277**

1. Introduction

Android operating system has become a popular mobile operating system. It is installed in millions of mobile devices and accounted for 61.9% OS market share in table unit sales to end users in 2013, while 36% for iOS and 2.1% for MS [1, 2]. The open nature of Android apps makes it a key target for malware authors. They can freely upload malicious apps to the Android app store or to a third-party alternative market. Any user is prone to infection through downloading and installing these malwares. A recent research [3] has shown that the malicious apps exist in both the official Android app store and unofficial markets with a rate of 0.02% and 0.2% respectively. According to Sophos security threat report in 2014, more than 300 Android malware families have been recorded in the period between August, 2010 and December, 2103 [4].

Android malwares continue to evolve, get smarter and more harmful. The harm's severity is ranged from showing Ads to launching DDoS attacks. Recently, malware authors start following technologies that have been applied in Windows. For example, some reports indicated a large-scale botnet controlling Android devices, namely Andr/GGSmart-A [5]. This botnet establishes a centralized control to instruct all of the Android devices it has infected to perform a task according to the desire of the author, i.e. send SMS messages that will be charged to the device owner. Ransomware is another harmful technology that attacks the Android devices and makes devices is inaccessible, and then it demands a payment to free it [6]. The trojanized malwares also have witnessed some evolution. The Trojan horse Andr/BBrodge-A, which is detected by Sophos, uses a privilege escalation exploit to install malicious app on the device. Sophos considered it as the most widely detected Android malware in 2013 [4]. A selection of malware harms along with examples on each one is mentioned in the following:

- Botnet activity harms, examples: Launching DDoS attacks; Sending premium rate SMS messages.
- Surveillance harms, examples: Audio; Camera; Call logs; Location.
- Data theft harms, examples: Account details; Contacts; Call logs; Phone number; Stealing data via App vulnerabilities.
- Financial harms, examples: Sending premium rate SMS messages; Stealing transaction authentication numbers; Extortion via ransomware; Making expensive calls.
- Impersonation harms, examples: SMS redirection; Sending email messages; Posting to social media.
- Annoying harms, examples: Showing adds.

All these threats have made the Android protection a thriving research area with considerable amount of still unsolved problems. The research of Android malwares detection has typically focused on platform based protection where the Android system is equipped with a mechanism of detection. The characterization of essential differences between malware and goodware apps is one of the research attempts [7]. The system of Android permissions represents a key in the running cycle of Android apps. The permission system is an important structure unit which is stored in the Android zipped archive (APK) manifest XML file. It grants or denies privileges to secure the user's privacy-relevant resources such as contacts, GPS, and SMS. Although these permissions are deemed to be useful in discriminating apps,

there have been limited attempts in the literature to exploit them in detecting malware apps [8]. The main challenge exists is the large overlap in the permissions. As the malicious apps tend to request a lot of permissions, most of these permissions are requested also by the goodware apps. Consequently, measurement and aggregation of the differences to obtain clear deviation between the two kinds of apps are too complicated. Moreover, the permission-based effective characterization of new variant of malwares, which are found almost every day, is another sustained challenge. With these challenges in mind, we design an intelligent approach for Android malware detection. Our system is built based on neuro-fuzzy inference methodologies. The contributions of our research are:

- Measuring the most significant permissions to discriminating optimally between the malware and the goodware apps.
- Combining most significant permissions into feature patterns to characterizing the Android apps efficiently.
- Building self-adaptive neuro-fuzzy inference system based on permissions requested to detect malware apps.
- Investigating the potentials of neuro-fuzzy inference systems in detecting Android malwares.

This paper is organized as follows: Section II presents an overview of the Android OS and reviews the literature. Section III introduces the methodology of the research. After that, Section IV presents and discusses the experimental results. Finally, we conclude our paper in Section V.

2. Background and Literature Review

In this section, we give an overview of the Android OS and existing Android malware defense techniques to motivate our new detection platform.

2.1 Android System Overview

This overview explores the Android OS in terms of the permission system. At the lowest level of the Android architecture, a customized Linux kernel is used to manage various system resources and hardware devices. Android apps run with a sandbox, which is a distinct system identity. Each app is assigned unique user ID at installation time and group IDs corresponding to requested permissions. According to the risks implied, the permissions are classified into four levels: 1) Normal, 2) Dangerous, 3) Signature, and 4) SignatureOrSystem. The Normal permissions are granted by the sandbox while the Dangerous permissions that might adversely impact the user environment are declared statically by the apps. The user will be prompted for approval at the install time. Permissions from the other two levels are granted according to certain conditions. Signature permissions are automatically granted when requesting application is signed with the same certificate of the app that declared the permissions, the SignatureOrSystem permissions are essentially limited to the apps that are pre-installed in Android's partition [9]. The grant process is not done dynamically; the Android's package installer performs it at the installation time either automatically as for normal permissions, or manually as for dangerous permissions. Information on the permissions requested can be extracted from the AndroidManifest.xml file. This file includes declaration tag (<uses-permission>) for each of the permissions requested. For example, an application that needs to monitor incoming SMS messages would specify:

```
<uses-permission android:name=  "android.permission.RECEIVE_SMS" />
```

It is accepted that the malicious apps demonstrate distinct malicious behaviors, and these behaviors resulting from specific permissions granted. Hence, malware apps often demonstrate distinct permissions. We specify here two examples of malicious behaviors: 1) collecting sensitive information and 2) obtaining financial gain. In the first behavior, a malware app should be granted unique permissions i.e.: READ_PHONE_STATE, SEND_SMS, and READ_OWNER_DATA. While in the second behavior, a malware app should be granted i.e: ACCESS_WIFI_STATE, READ_PHONE_STATE, and INTERNET permissions.

2.2 Android Protection

The protection strategies are grouped into two sets: (i) marketplace protection and (ii) platform protection. The marketplace protection is performed by the marketplace own tools. For example: analyzing submitted apps before publishing them; forcing the authors to register their apps allowing them to claim authorship. However, these actions have proven insufficient to prevent propagation of malware since the manual review and authorization of apps is a difficult task to perform with an enormous number of apps being submitted every day. As an alternative, automated approaches have been recently published. Google announced Google Android's Bouncer to automatically authorize submitted apps [10]. Another example is the DroidRanger for detecting smartphone malware in Android markets [3]. The main critique against this type of protection is that even if the app review processes, manual or automatic, are perfect, it cannot defend malwares uploaded to unofficial markets in which there are no guaranties about the trustworthiness.

In the platform protection set, the Android platform incorporates a technique to detect the actuation of malicious apps once installed in the device. A number of classification techniques adopted to detect malware apps have been introduced in the literature. P. Faruki et. al. [11] divide the techniques into: static, dynamic, and hybrid according to how code is analyzed to detect the malware apps. In the static techniques, the apps code is analyzed off-line, without running it, while the code is monitored dynamically to inspect the interaction with the system in the dynamic techniques. The hybrid techniques leverage the good of both the static and dynamic techniques. The authors in [12] introduced a taxonomy of the malware detection techniques in terms of monitoring, analysis, and identification. In this paper, we divide the protection techniques into three subsets: (i) signature-based, (ii) anomaly-based, and (iii) permission-based. In the subsequent sections, we review the pros and cons and give some examples of research published for each subset.

A. Signature-based Technique:

The main benefit of signature-based detection techniques lies in its accuracy of detection of known malware and simplicity of implementation. The existing commercial anti-malware systems use signature-based detection methods. They analyze malware codes off-line and extract features to create a unique signature for the malware code. Hence, for each malware, signature-based systems are equipped with at least one signature [13], and usually, the signatures are stored in accompany database. In this regards, these systems can detect only malware for which a signature is available, and they fail against zero-day and polymorphic malwares. The signature generation is done manually and it is a difficult and time-consuming task. Also it is impossible to perform in full extent due to the massive number of applications. All of these factors can come up with negative results. Zhou et al. [7] reported that common smartphone antivirus apps detect only between 20.2% and 79.6% of analyzed malware.

Many applications could be listed under this subset, such as the AndroSimilar [14] which is a robust statistical feature signature-based method to detect Android malware apps, and the AppGuard [15] which is a malware prevention system that monitors the application inline.

B. Anomaly-based Technique:

Several works have been introduced to detect Android malwares by analyzing dynamically various features that serve to characterize anomaly behaviors. These techniques monitor constantly system calls, network activities, event logs, user activity permissions, and program traces. Although anomaly-based techniques are arguably more powerful than the signature-based techniques in terms of polymorphic and zero-day malware detection, they cause big resources consumption, particularly if a large number of information is collected directly over a running app.

The dominant features of what has been published in this subset are: the real-time monitoring and the use of machine learning algorithms. In [16], Shabtai et al. proposed a light-weight Android malware detection system called Andromaly. This system is a real-time monitoring system that collects various system metrics, such as CPU usage, amount of data transferred through network, number of active processes and battery usage. The collected features are classified using some of machine learning methods like: K-Means, Decision Trees, Bayesian Networks, and Naïve Bayes. Schmidt et al. [17] proposed a system for both Android and Symbian operating systems. Initially, the app's function calls are extracted and the data is analyzed using machine learning method, the Decision Trees. Secondly, the system monitors more features such as free RAM memory, CPU usage, SMS count for further analyzing behavior. Classification is done in the cloud using machine learning algorithms such as Support Vector Machines (SVM), and Tree Kernels. MADAM system [18] periodically monitors a number of features to model app behavior and classify it as goodware or malware. The collected observations are classified using K-Nearest Neighbor (K-NN). Finally, TStructDroid [19] presents a real-time malware detection system. The proposed system monitors Process Control Blocks (PCB) and analyzes the time-series, feature logging, and frequency component analysis of data. It uses machine learning method to analyze monitored data and classify apps into goodware and malware.

C. Permission-based Technique:

Malware detection of Android apps based on permissions is rather new research field. The main goal of this research is to characterize the fundamental differences between the goodware apps and the malware apps in term of permissions. Usually, an Android app is represented by a vector of binary values, each of which is associated with one of the 130 Android official permissions [8]. In the vector, the value of "1" indicates that the permissions are required by the apps and the value of "0" indicates that the permission is not required. Since the dataset size does affect the accuracy of the detection, collecting big data for training the system is one of the considerable factors. Although each set of permissions implies certain differences between a goodware app and a malware app, the overlapping between these permissions over the apps makes it too difficult to obtain a comprehensive insight on the deviation between the two classes. This pushed the researchers to use sophisticated methodologies such as machine learning. For example, the authors in [20] introduced a feature learning framework using SVM, Decision Trees, and Bagging machine learning methods. The system proposed by Sanz et. al. [21] is based on the permissions requested in Androidmanifest.xml using Naïve Bayes, Random Forest, and J48 machine learning. Another example introduced by C.-Y Huang et. al. [22] to map the requested permissions in the Linux kernel. The mapped attributes are used

with AdaBoost, Naïve Bayes, Decision Trees, and SVM machine learning methods to identify the malware apps. The research proposed by Shalaginov and Franke [23] is another example that employed permissions to detect Android malwares. Each permission is assigned a degree security risk from 0 to 4. The higher degree is assigned to the most dangerous permission. Neuro-fuzzy approach is applied to classify mobile applications into benign and malicious. The system achieved reliable classification accuracy of 76%.

With the use of machine learning methods, intuitively, it is a difficult to draw a decision boundary between malware apps and goodwares apps based on highly overlapped permissions. This problem calls for a precise, expert modeling system that focuses on accuracy. We believe that the fuzzy logic methodologies are useful in the development of expert modeling systems as these methodologies are successfully tested to solve many problems in many research areas such as estimation [24], classification [25], and recognition [26].

3. Research Methodology

As this paper is intended to investigate the feasibility of neuro-fuzzy techniques to Android protection systems, we only applied simulation approach. Simulation provided apportunities for finiding adequate parameters, simulating the potential changes to the system, and predicating the changes impact on the system. The malware detection system is introduced in this section. Firstly, in the subsequent sections the k-ANFIS structure is introduced. Then, the dataset employed in this research is detailed. At the end, the operations of feature selection and feature pattern construction are explained.

3.1 Android Malware Detection System, k-ANFIS

The name “k-ANFIS” is acronym of kEFCM-based Adaptive Neuro-Fuzzy Inference System. k-ANFIS is a combination of ANFIS [27] and kEFCM [28]. ANFIS is a neuro-fuzzy model that combines fuzzy inference systems and artificial neural networks while kEFCM is kNN-based evolving fuzzy clustering method. The architecture of k-ANFIS is shown in **Fig. 1**. k-ANFIS is the first neuro-fuzzy inference system that employs kEFCM to create fuzzy rules. kEFCM brings various enticing advantages over existing methods, including: 1) reducing the complexity of computation, 2) on-line clustering, and 3) fuzzy evolving. Regardless of the number of neighbors involved in the clustering which is can be precisely determined, k-ANFIS does not use any tuning parameters that could affect the overall performance. The adaptive network of ANFIS is a multilayer feed-forward network. Each neuron in a layer performs a particular function on its input signals and transmits its output to neuron(s) in the next layer. There are two types of neurons, adaptive and fixed. Adaptive neurons have parameters while fixed neurons have no parameters.

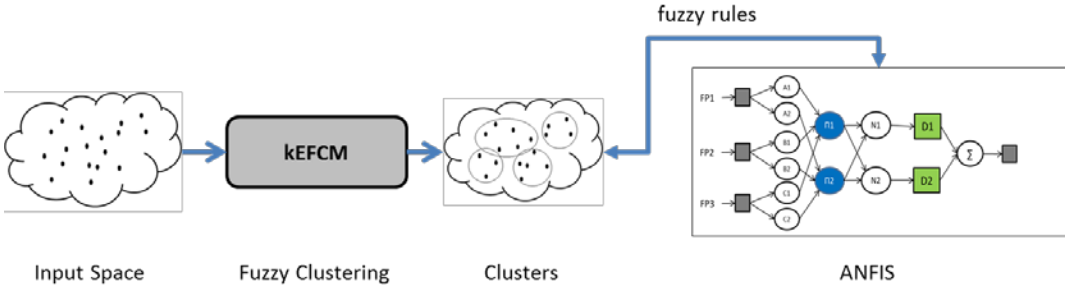


Fig. 1. k-ANFIS structure

Fig. 2 illustrates the architecture of ANFIS, where the squares represent adaptive neurons and circles represent fixed neurons.

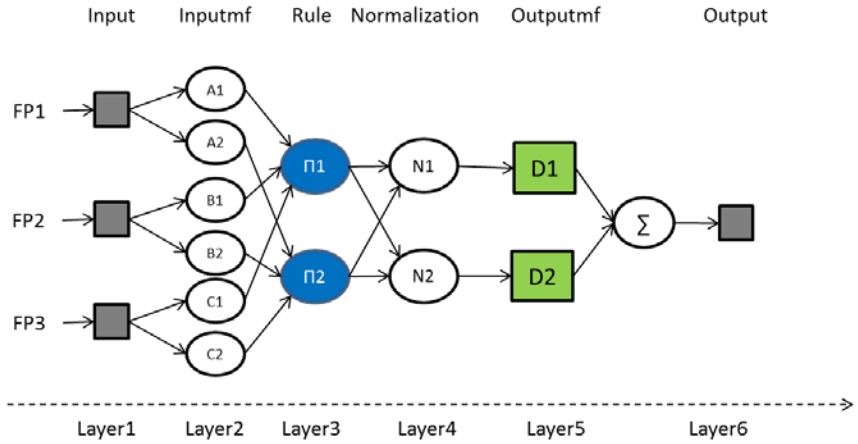


Fig. 2. ANFIS architecture

ANFIS constructs a fuzzy inference system corresponding to a dataset of input/output. ANFIS uses in most cases a hybrid system of two learning methods, back-propagation and least square type to adjust the membership function parameters of the fuzzy inference system. The parameters related to the membership function are modified through the learning process.

ANFIS uses Takagi-Sugeno, a method of fuzzy 'if-then' inference rules. The if-part of the rule is called the antecedent, while the then-part of the rule is called the consequent. A typical Takagi-Sugeno rule is as follows:

$$IF\ x_1 is\ A_1 \& x_2 is\ A_2 \ldots . \& x_n is\ A_n \quad THEN\ y = f(x_1, x_2, \ldots . x_n) \ldots \ldots . . (1)$$

Where x_1, x_2, and x_n: input variables and A_1, A_2, and A_n: fuzzy sets obtained by applying a membership function, which defines how each input point is mapped to a membership value between 0 and 1. There are several types of membership functions i.e. Triangular, Trapezoidal, and Gaussian. Choosing the membership function depends on the problem at hand. If y is a constant, then the Takagi-Sugeno is a zero-order Sugeno type, and it is said to be a first-order fuzzy type if y is a first-order polynomial:

$$y = k_0 + k_1 x_1 + k_2 x_2 + \cdots + k_n x_n \ldots \ldots \ldots \ldots (2)$$

As shown in **Fig. 2**, ANFIS architecture comprises 6 layers. Each layer involves a number of neurons and performs a specific task.

Layer 1, the input layer: neurons of this layer make no computations. They pass the input data to Layer 2.

Layer 2, the fuzzification layer: neurons in this layer use a fuzzy membership function to compute the membership degrees of the inputs in the corresponding fuzzy sets. Generally, a fuzzy membership function depends on three parameter sets and described as a bell-shape like. For example, the Triangular membership functions μ of a vector x, depends on three parameters a, b, and c, as given by:

$$\mu(x,a,b,c) = max\left(min\left(\frac{x-a}{b-a},\frac{c-x}{c-b}\right),0\right) \quad \text{.....................................(3)}$$

Where the parameters a and c locate the "feet" of the triangle and the parameter b locates the peak. **Fig. 3** shows the triangle-shaped membership function.

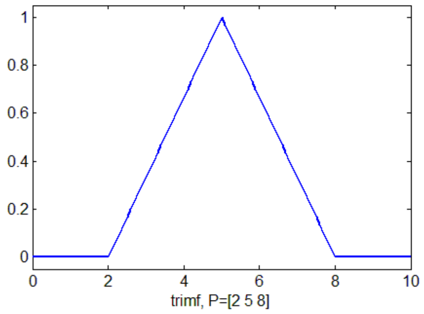


Fig. 3. Triangle-shaped membership function

Layer 3, the rule layer: each neuron in this layer represents a single Takagi-Sugeno fuzzy rule. Each neuron receives inputs from the respective fuzzification neurons in Layer 2 and combines them through a specific t-norm operator, which is usually multiplication to get the firing strength w_i of the rule r_i:

$$w_i = t\left(\mu_i(x_{1i}), \mu_i(x_{2i})\right) = \mu_i(x_{1i}).\mu_i(x_{2i}) \quad \text{........................(4)}$$

Where $\mu_i(x_{1i})$ and $\mu_i(x_{2i})$ represent values of the fuzzy function membership for x_{1i} and x_{2i} input variables.

Layer 4, the normalization layer: neurons in this layer compute the normalized firing strength of a given rule $\overline{w}_i$ by dividing each rule firing strength w_i by the summation of all of the rules, as follows:

$$\overline{w}_i = \frac{w_i}{\sum_{j=1}^{n} w_j} \quad \text{...(5)}$$

Layer 5, the defuzzification layer: Neurons in this layer are called defuzzification neuron. Each defuzzification neuron calculates the weighted consequent value of a given rule as:

$$y_i = \overline{w}_i.f_i \quad \text{...(6)}$$

$$y_i = \overline{w}_i.[k_{0i} + k_{1i}x_{1i} + k_{2i}x_{2i} + \cdots + k_{ni}x_{ni}] \quad \text{...(7)}$$

Where x_{1i} , x_{2i}, … x_{ni} are the input, y_i is the output of the defuzzification rule i, and $k_{0i}, k_{1i}, k_{2i,}$and k_{ni} are the consequent parameters of rule i.

Layer 6, the output layer: This layer has only one neuron which performs the aggregation of all incoming layer 5's outputs. Usually, the aggregation process is summation. The global output of the system is given by [29]:

$$o_i = \sum_{i=1}^{n} y_i \qquad \text{..... (8)}$$

Fuzzy rule generation is the most important task in modeling neuro-fuzzy inference systems. One of the common methods to accomplish this task is the use of fuzzy clustering-based methods. Such methods decompose the input space into a certain number of clusters, where each input data belongs to a cluster in a certain degree. Every cluster is assigned to a specific Takagi-Sugeno fuzzy rule. The rules resulted are more tailored to the input data than they are generated without clustering.

k-ANFIS uses the evolving fuzzy clustering method, the kEFCM, which is an evolving fuzzy clustering method that overcomes the problems of computational cost and clustering complexity diagnosed in the traditional version. kEFCM process divide into two task. First task orders all the points in such a way that the points in the same cluster are more similar to each other than to those in other clusters. Geometrically, the clusters are circles. The least-squares method is used in determining the circle center and radius. The membership degree of a point in a cluster is identified by the Euclidean distance between the point and the cluster center. This task is repeated on the remaining data points. As the first task may create unwanted overlapped clusters, the main function of the second task is to apply an optimization procedure that handles two constraints: (i) The number of clusters that contain small cluster(s) is equal to 0; (ii) The number of clusters that include points less than k is equal to 0.

3.2 Dataset

For the purpose of evaluation, we used the dataset of Genome project [30]. The main advantage of this dataset is that it is almost free of noise, which led us to not considering type-2 fuzzy system. Type-2 fuzzy system is used where it is difficult to determine an exact membership function such as when the training data are corrupted by noise [31]. The dataset consists of more than 1,200 malware samples that cover the majority of existing Android malware families collected between August 2010 and October 2011. The researchers released the entire dataset to help the security community in developing efficient malware detection techniques [7]. However, the Genome data actually contains only malware samples. To make our dataset more comprehensive and up to date, we downloded goodware apps from public Android app store, Google Play. The goodware apps are analyzed in terms of permissions and added to the dataset. From this large collection of Android malwares, we selected a total of 200 samples, 100 malwares and 100 goodwares. For each sample, there are 50 features. Each feature refers to one request of permission, for example, read_phone_state, access_network_state, sms_received and sig_str. The features share same binary values where 1 means the malware possesses the feature and 0 means the malware does not possess it.

3.3 Features Selection

When the number of extracted permission based features is large, some of which may be redundant or irrelevant, and this introduce several problems such as misleading the detection. This paper uses the Information Gain Ratio method (IGR) to select the most significant features of Android malware. Information Gain Ratio method works based on the extraction of similarities between sets of Android app and then it provides the greatest weight to the most effective features based on the class of goodware and malware apps belonging to IGR, as explained in the following equations:

$$gain\_r(X,C) = \frac{gain(X,C)}{split\_info(C)} \quad \text{..... (9)}$$

$$split\_info(C) = \sum_i \left(\frac{|C_i|}{|C|}\right) log \frac{|C_i|}{C} \quad \text{..... (10)}$$

Where, gain_r (X,C) represents the gain ratio of the feature X frequency in class C. Ci and |Ci| denote the frequency of features X in class C, the ith subclass of C and the number of features in Ci, respectively. The features that have been selected along with the ranks resulted from the IGR are shown in **Fig. 4**.

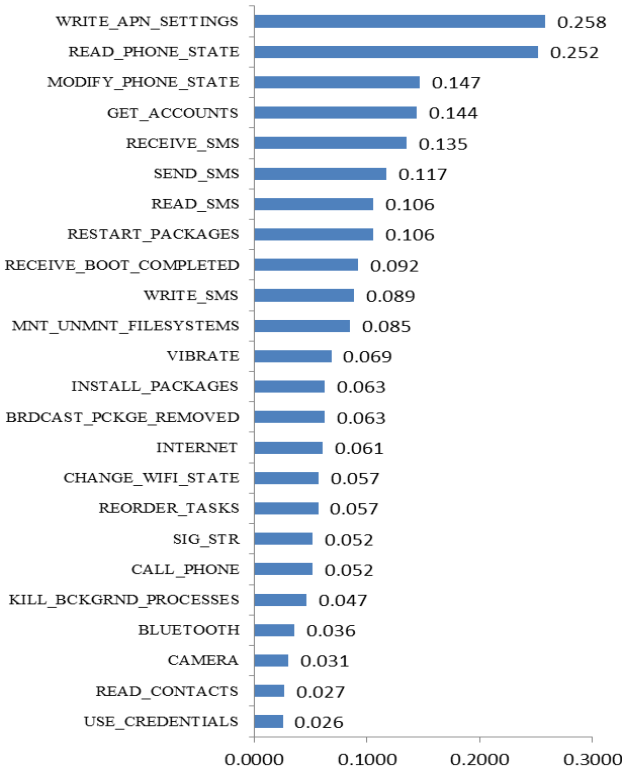


Fig. 4. The selected Android malware features

3.4 Feature Pattern Construction

Upon identifying the most significant features of Android samples, we started the process of feature pattern (FP) construction. As mentioned before, each feature value is a binary data type. For instance, the feature: "WRITE_APN_SETTINGS" will store 1 if the malware write access point name (APN) settings, otherwise will store 0. In addition to aviding CPU overload while execution the detection algorithm with a lot of features, the construction process aimed at isolating goodware and malware samples perfectly by selecting the largest number of features that gain high ranks. The number of features selected was 24 features. Based on the values (0 or 1), we considered each feature as bit. Thereby, the number of features is devided into three group (bytes) which formed the system training and testing input. The FP construction process for each sample involved the following steps:

1. Selecting the top 24 features based on the results of the IGR method,
2. Ordering the features according to the IGR ranks,

3. Dividing the 24 feature values into 3 string of binary values or bytes, and finally
4. Converting the bytes to decimal number, each number represents one FP.

Example of feature patterns for 11 Android malwares can be seen in **Table 1**. The last column is the class of the malware, 1 means malware and 0 means goodware.

Table 1. Examples of the feature patterns (FP)

Malware#	FP(1)	FP(2)	FP(3)	Class
1	96	77	135	1
2	0	65	7	0
3	64	119	247	1
4	0	67	114	0
5	64	119	247	1
6	65	203	246	1
7	0	73	10	0
8	65	73	2	1
9	32	64	0	1
10	0	64	2	0
11	0	65	0	0

4 Results and Discussion

All the 200 Android samples in the dataset are arranged for 3-input-1-output system as this format: (FP1, FP2, FP3, C), where FP1, FP2, and FP3 are the input values and C is the output variable. Before approaching the experiments, we tuned kEFCM by testing several values of k, the number of neighbors involved in the classification. The test was on randomly selected 20 samples. Finally and based on the results, the k=7 is selecetd as it gave the highest accuracy [32]. Typically, an initial step is to split the dataset into training part and testing part. We use the n-fold cross-validation technique to perform the split step [33]. We choose (n=10) since the 10 value seems to be an optimal number of folds that optimizes the time it takes to complete the test [34]. The dataset is randomly split into 10 mutually exclusive sub-datasets of equal size. The k-ANFIS is trained 10 times. Each time, we use 9 folds for the training and leave one single fold for the testing. Typically, the overall cross-validation accuracy is affected by the places and number of the classes in the 10 folds. To assure results with lower bias and lower variance, we performed the process of stratification [35], which aims to create folds in a way that they contain the same proportion of classes (malware and goodware). **Table 2** shows number of malware and goodware samples along with the total number of samples in each fold.

Table 2. 10-folds cross-validation and stratification

	Malware samples	Goodware samples	Total
Training	90	90	180
Testing	10	10	20
Total # of samples in each fold			200

The overall accuracy of cross-validation is calculated by averaging the n individual accuracy measures:

$$CVA = \frac{1}{n}\sum_{i=1}^{n} A_i \qquad \text{..... (11)}$$

Where CVA is the cross-validation accuracy, n is the number of folds used, and A_i is the accuracy measure of each fold. We used different accuracy measures for each experiment conducted.

Experiment-1: Definition of Membership Function

One of the most significant steps in developing a neuro-fuzzy inference system is defining fuzzy membership function MF [36]. We tested the performance of k-ANFIS using five MFs: two Gaussian membership functions: gauss and gauss2, sigmoid member function: sig, generalized bell membership function: gbell, and trapezoidal membership function: trap. The main objective of using these MFs is to determine which kind of MFs is important in our approach i.e. straight line, curve, or asymmetric. We established the Variance Account For (VAF) as a CVA to assess the performance of k-ANFIS on each MF. The VAF is calculated for each fold as follows:

$$VAF = \left(1 - \frac{var(M - P)}{var(P)}\right).100 \qquad \text{..... (12)}$$

Where the variance (var) in set (y): $var(y) = \frac{1}{n}\sum_{i=1}^{n} P_i$

A detection model is said to be excellent when the VAF is 100%. The MF parameters and types along with CVA achieved based on VAF are shown in **Table 3**. The success of the MFs was different regarding the VAF. The lowest CAV value is achieved by Trap MF, 76.87. The most desired result is given by Sig MF, 78.75. Accondingly, the Sig MF will be adopted in the next experiments. On the otherhand, all remaining MFs exhibited approximatelly the similar performances. Therefore, the final conclusion of experiment-1 is that k-ANFIS approach needs to specify asymmetric membership function.

Table 3. k-ANFIS accuracy by five different MFs

MF	# of Param.	Formed using	CVA based on VAF
Gauss	3	Curve	77.11
Gauss2	2	Curve	78.30
Sig	2	Asymmetric and close curve (i.e. not open to the left or right)	78.75
Gbell	2	Curve	78.45
Trap	4	Straight lines	76.87

Experiment-2: Comparison of Error Measurements

Neuro-fuzzy inference binary classification systems usually produce continuous output score in the range of 0 .. 1, where 0 is the foreground class and 1 is the background class. In our case, 0 is the goodware class and 1 is malware class. The classification could be considered as regression problem where a classification system is accepted as excellent when the difference between the score implied by it and the observed class is equal to zero. In this round of experiment, we used four regression metrics to calculate the differences and estimate the CVA, as follows:

1. The root mean square error (RMSE): to quantifies the difference between values implied by the system and the observed values:

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(M_i - P_i)^2} \quad \text{..... (13)}$$

2. The normalized mean square error (NMSE): to estimate the overall deviations between values implied by system and the observed values:

$$NMSE = \frac{1}{n}\sum_{i=1}^{n}\frac{(M_i - P_i)^2}{\bar{M}.\bar{P}} \quad \text{..... (14)}$$

$$\bar{M} = \frac{1}{n}\sum_{i=1}^{n} M_i$$

$$\bar{P} = \frac{1}{n}\sum_{i=1}^{n} P_i$$

3. The mean absolute error (MAE): to measure how close the values implied by the system to the observed values:

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|P_i - M_i| \quad \text{..... (15)}$$

4. The symmetric mean absolute percentage error (SMAPE): to measure the accuracy of the system based on relative errors:

$$SMAPE = \frac{1}{n}\sum_{i=1}^{n}\frac{|M_i - P_i|}{|M_i + P_i|} \quad \text{..... (16)}$$

Where P_i refers to the values implied by the system and M_i refers to the observed values.

To demonstrate the potentials of k-ANFIS, the ANFIS system plus another well-known system, DENFIS [37] are benchmarked. Three reasons behind the selection of these systems: (1) they improve efficiency in variety of different detection tasks and (2) they are applicable to the field of malware detection, and (3) these two systems are resemble k-ANFIS with regrads to fuzzy rules type. Given DENFIS system, the first-order Takagi-Sugeno type fuzzy rules are employed where the linear functions in the consequence part are created and updated by linear-square estimator [37]. This type of fuzzy rules can be expressed as in equations 1 & 2. **Table 4** shows the performance of k-ANFIS, ANFIS and DENFIS for the dataset, the performance of the models regarding to classification errors are compared. An observation can be made which is that all regression merics values indicate that testing folds of k-ANFIS possess the smallest values compare to ANFIS and DENFIS. **Fig. 5** shows comparison between the values of CVA of the systems. Based on the figure, it is can be clearly seen that the results from k-ANFIS are significantly the better. For example, the CVA of RMSE value of k-ANFIS is 0.3495 which is slightly small when comparing to CVA values of ANFIS and DENFIS which are 0.3496 and 0.3792 respectively. Despite that k-ANFIS and ANFIS show comparable values of CVA of RMSE, NMSE, MAE, and SMAPE, k-ANFIS shows the smallest values. Futhermore, the CVA values of these metrics of k-ANFIS are much less than

DENFIS CVA values. The final verdict is that the k-ANFIS system outperforms ANFIS and DENFIS.

Experiment-3: Comparison of Accuracy

From the binary classification evaluation perspective, there is a need of separating the continuous output score produced by a neuro-fuzzy inference binary classification system into two classes as clear implied by a hard-labeling, where each output value either belongs to the foreground class or belongs to the background class [38]. We separated the continuous output values by deciding on cutoffs (thresholds) in the range of 0...1, where the malware output values are on one side of the cutoff and the goodware output values are on the other side of the cutoffs.

Four cutoffs have been decided in this experiment: 0.10, 0.25, 0.35, and 0.40. For each cutoff, we computed the confusion matrix, accuracy, error, precision, and recall, which are common measures of binary classification performance. The confusion matrix for our classification problem that has two possible outcomes: 0 (goodware) and 1 (malware) is depicted as follows:

		Actual	
		1	0
Classified	1	TP	FP
	0	FN	TN

Where TP: malware and is classified as malware; FP: malware and is classified as goodware; FN: goodware and is classified as malware; TN: goodware and is classified as goodware.

Table 4. Classification performance of k-ANFIS, ANFIS, and DENFIS

Fold#	RMSE	NMSE	MAE	SMAPE
System: k-ANFIS				
1	0.251	0.213	0.219	0.549
2	0.289	0.289	0.244	0.550
3	0.253	0.230	0.210	0.542
4	0.417	0.696	0.351	0.635
5	0.352	0.493	0.302	0.582
6	0.250	0.218	0.209	0.530
7	0.393	0.590	0.335	0.629
8	0.395	0.594	0.352	0.654
9	0.448	0.742	0.415	0.680
10	0.447	0.765	0.409	0.680
System: ANFIS				
1	0.251	0.213	0.219	0.549
2	0.289	0.289	0.244	0.551
3	0.253	0.230	0.210	0.542
4	0.417	0.697	0.351	0.635
5	0.352	0.493	0.302	0.582
6	0.250	0.219	0.209	0.530
7	0.393	0.590	0.335	0.629
8	0.395	0.594	0.352	0.654
9	0.448	0.742	0.415	0.680
10	0.448	0.765	0.409	0.680

System: DENFIS				
1	0.281	0.243	0.245	0.578
2	0.323	0.313	0.260	0.581
3	0.285	0.250	0.227	0.567
4	0.447	0.719	0.373	0.657
5	0.374	0.503	0.312	0.626
6	0.287	0.253	0.229	0.564
7	0.428	0.621	0.377	0.658
8	0.413	0.615	0.391	0.686
9	0.470	0.783	0.425	0.720
10	0.484	0.782	0.441	0.699

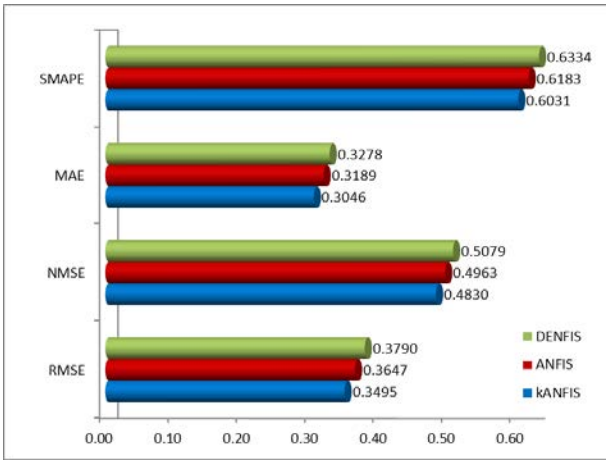


Fig. 5. CVA in terms of regression metrics

Classification accuracy:

$$accuracy = \frac{(TP + TN)}{(TP + FN + TN + FP)} \ldots\ldots\ldots\ldots (17)$$

Error rate:

$$error = \frac{(FP + FN)}{(TP + FN + TN + FP)} \ldots\ldots\ldots\ldots (18)$$

Precision: proportion of classified malwares which are actual malwares:

$$precision = \frac{TP}{(TP + FP)} \ldots\ldots\ldots\ldots (19)$$

Recall: proportion of actual malwares which are classified malwares:

$$recall = \frac{TP}{(TP + FN)} \ldots\ldots\ldots\ldots (20)$$

Table 5 shows the rate of accuracy, error, precision, and recall achieved by k-ANFIS, ANFIS, and DENFIS for each cutoff. Whilst the results from all systems may be satsifactory in terms of accuracy, k-ANFIS system is better for all cutoffs comparatively. The ANFIS accuracy of all cuttoffs is very close to the accuracies obtained by DENFIS. In terms of how well the system classifies malware and goodware samples, we can see that k-ANFIS due to precision and recall measures gives better results than ANFIS and DENFIS for all cutoffs.

Table 5. The classification accuracy for each cutoff

Cutoff	Accuracy	Error	Precision	Recall
System: k-ANFIS				
0.10	20%	80%	21%	16%
0.25	48%	52%	36%	44%
0.35	62%	38%	54%	60%
0.40	75%	25%	75%	66%
System: ANFIS				
0.10	15%	85%	13%	14%
0.25	45%	55%	38%	47%
0.35	62%	38%	56%	57%
0.40	70%	30%	68%	65%
System: DENFIS				
0.10	13%	87%	11%	13%
0.25	43%	57%	34%	45%
0.35	61%	39%	57%	56%
0.40	70%	30%	70%	64%

The next comparison focuses on the false alarm rates. The Receiver Operating Characteristics (ROC) is commonly used to compare classifiers in terms of FP rates and TP rates. It is a two-dimensional graph in which TP rate is plotted on the Y axis and FP rate is plotted on the X axis. To compare classifiers, an important measure of the accuracy is used, the area under the ROC curve (AUC) [39]. The value of AUC is between 0.5 and 1. If the AUC is equal to 1.0 then the classifier is 100% accurate because the both the TP rate and FP rate are 1.0, so there is no FP and no FN produced. **Fig. 6, 7**, and **8** depict the ROC curves for k-ANFIS, ANFIS, and DENFIS classifiers. The results confirm that the DENFIS's area under the curve (AUC=0.9329) is slightly less than both the k-ANFIS and ANFIS (AUC=0.9333). However, all classifiers achieved high AUC, more than 0.9, meaning that they suffer from very low number of false alarms. From the results, despite the equal of AUC value for k-ANFIS and ANFIS, k-ANFIS performs generally better than ANFIS at the region FP rate between 0.25 - 0.10 and between 0.175 - 0.20 of ROC space and that gives k-ANFIS slight advantage.

Summary: the results confirm the suitability of neuro-fuzzy inference systems detecting Android malwares, with encouraging results:

- **Definition of MF:** testing the performance of k-ANFIS using five different MFs yielded that the Sig MF clearly outperforms gauss, gauss2, sigmoid, gbell, and trap MFs.
- **High detection accuracy:** the accuracy of the systems ranging from 70% - 75% for the cutoff 0.4, which is below the mid rage (0.5). However, k-ANFIS achieved only a slightly more accuracy rate than the other systems.
- **False alarm rates:** Receiver operator characteristics (ROC) graphs confirm that the false alarm rates for all of the systems is low, which means a competitiveness between the systems.

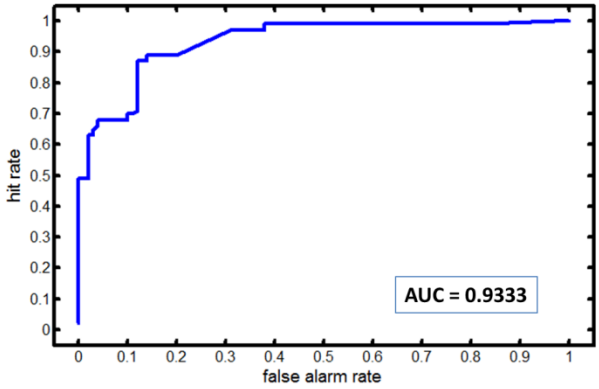


Fig. 6. ROC for classification by k-ANFIS

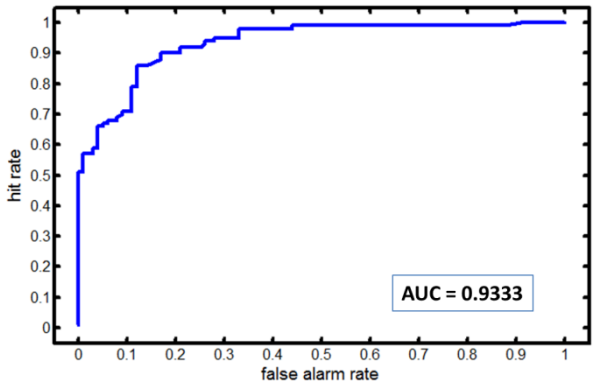


Fig. 7. ROC for classification by ANFIS

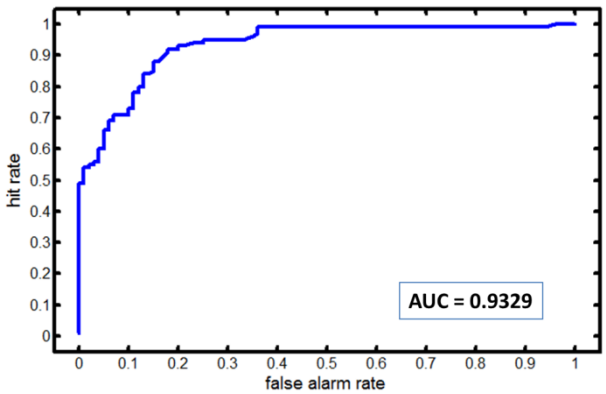


Fig. 8. ROC for classification by DENFIS

5 Conclusion

An intelligent approach for malware detection in Android by using 24 features is presented. The usual way to detect malware attacks is by adopting directly a set of features and distinguishing the malware from the goodware software. In our case, this way is not applicable since that we have a lot of malware features, and this will cause CPU overload while execution

the app. Our methodology nominates the 24 binary value features based on the IGR algorithm and encapsulates them into 3-byte feature pattern (FP). We developed a new intelligent system, k-ANFIS that possesses same features of ANFIS system and uses a new clustering methodology. k-ANFIS detects the type of Android apps by classifying the FP into malware or goodware. The evaluation of the system involved selection of the MF and comparing k-ANFIS performance against performance of ANFIS and DENFIS. The overall results have shown high performance, which summarizes the suitability of neuro-fuzzy inference system for detecting Android malware based on system permissions.

References

[1] L.-K. Yan and H. Yin, "DroidScope: Seamlessly Reconstructing the OS and Dalvik Semantic Views for Dynamic Android Malware Analysis," *USENIX Security Symposium*, pp. 569-584, 2012.

[2] Gartner, “Worldwide Tablet Sales Grew 68 Percent in 2013, With Android Capturing 62 Percent of the Market,” 2014. http://www.gartner.com/newsroom/id/2674215

[3] Y. Zhou, Z. Wang, W. Zhou, and X. Jiang, "Hey, You, Get Off of My Market: Detecting Malicious Apps in Official and Alternative Android Markets," in *Proc. of the 19th Network and Distributed System Security Symposium (NDSS)*, 2012.

[4] Sophos Security Threat Report 2014, 2014. http://www.sophos.com/en-us/threat-center/security-threat-report.aspx

[5] L. Patel and D. Sharma, "CYBER TRIANGLE," *International Journal For Technological Research In Engineering*, vol. 1, pp. 799-807, 2014.

[6] T. T. Gotora, K. Zvarevashe, and P. Nandan, "A Survey on the Security Fight against Ransomware and Trojans in Android," *International Journal of Innovative Research in Computer and Communication Engineering*, vol. 2, pp. 4115-4123, 2014.

[7] Y. Zhou and X. Jiang, "Dissecting android malware: Characterization and evolution," in Security and Privacy, *2012 IEEE Symposium on*, pp. 95-109, 2012. Article (CrossRef Link)

[8] P. Xiong, X. Wang, W. Niu, T. Zhu, and G. Li, "Android malware detection with contrasting permission patterns," *Communications*, vol. 11, pp. 1-14, 2014. Article (CrossRef Link)

[9] W. Xu, F. Zhang, and S. Zhu, "Permlyzer: Analyzing permission usage in Android applications," *Software Reliability Engineering (ISSRE)*, 2013 IEEE 24th International Symposium on, pp. 400-410, 2013. Article (CrossRef Link)

[10] S. Mansfield-Devine, "Android malware and mitigations," *Network Security*, vol. 2012, pp. 12-20, 2012. Article (CrossRef Link)

[11] P. Faruki, A. Bharmal, V. Laxmi, V. Ganmoor, M. Gaur, M. Conti, and R. Muttukrishnan, "Android Security: A Survey of Issues, Malware Penetration and Defenses," *Communications Surveys & Tutorials, IEEE*, vol. PP, pp. 1-1, 2015.

[12] G. Suarez-Tangil, J. Tapiador, P. Peris-Lopez, and A. Ribagorda, "Evolution, detection and analysis of malware for smart devices," 2013.

[13] R. Fedler, J. Schütte, and M. Kulicke, "On the effectiveness of malware protection on Android," *Tech Repo April Fraunhofer AISEC*, 2013.

[14] P. Faruki, V. Laxmi, A. Bharmal, M. Gaur, and V. Ganmoor, "AndroSimilar: Robust signature for detecting variants of Android malware," *Journal of Information Security and Applications*, 2014.

[15] S. G. M. Backes, C. Hammer, M. Maffei, and P. Styp Rekowsky, “Appguard-realtime policy enforcement for thirdparty applications,” 2012. http://scidok.sulb.uni-saarland.de/volltexte/2012/4902

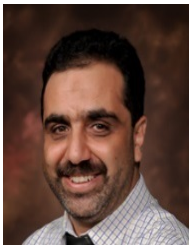
[16] A. Shabtai, U. Kanonov, Y. Elovici, C. Glezer, and Y. Weiss, ""Andromaly": a behavioral malware detection framework for android devices," *Journal of Intelligent Information Systems*, vol. 38, pp. 161-190, 2012. Article (CrossRef Link)

[17] A.-D. Schmidt, "Detection of Smartphone Malware," *Berlin Institute of Technology*, 2011.

[18] G. Dini, F. Martinelli, A. Saracino, and D. Sgandurra, "Madam: a multi-level anomaly detector for android malware," *in Computer Network Security, ed: Springer*, pp. 240-253, 2012. Article (CrossRef Link)
[19] F. Shahzad, M. Akbar, S. Khan, and M. Farooq, "Tstructdroid: Realtime malware detection using in-execution dynamic analysis of kernel process control blocks on android," National University of Computer & Emerging Sciences, *Tech. Rep*, 2013.
[20] N. Peiravian and X. Zhu, "Machine Learning for Android Malware Detection Using Permission and API Calls," in *Proc. of Tools with Artificial Intelligence (ICTAI), 2013 IEEE 25th International Conference on*, 2013, pp. 300-305. Article (CrossRef Link)
[21] B. Sanz, I. Santos, C. Laorden, X. Ugarte-Pedrero, P. G. Bringas, and G. Álvarez, "Puma: Permission usage to detect malware in android," in *proc. of International Joint Conference CISIS'12-ICEUTE´ 12-SOCO´ 12 Special Sessions*, pp. 289-298, 2013. Article (CrossRef Link)
[22] C.-Y. Huang, Y.-T. Tsai, and C.-H. Hsu, "Performance Evaluation on Permission-Based Detection for Android Malware," *Advances in Intelligent Systems and Applications-Volume 2, ed: Springer*, pp. 111-120, 2013. Article (CrossRef Link)
[23] A. Shalaginov and K. Franke, "Automatic rule-mining for malware detection employing Neuro-Fuzzy Approach," *Norsk informasjonssikkerhetskonferanse (NISK)*, vol. 2013, 2014.
[24] T. Sumithira, A. Nirmal Kumar, and R. Ramesh Kumar, "An adaptive neuro-fuzzy inference system (ANFIS) based Prediction of Solar Radiation: A Case study," *Journal of Applied Sciences Research*, vol. 8, 2012.
[25] M. Watts and N. Kasabov, "Simple evolving connectionist systems and experiments on isolated phoneme recognition," *Combinations of Evolutionary Computation and Neural Networks*, 2000 IEEE Symposium on, pp. 232-239, 2000. Article (CrossRef Link)
[26] A. Ghobakhlou and N. Kasabov, "A methodology for adaptive speech recognition systems and a development environment," in *Proc. of Artif. Neural Netw. Neural Inform. Process*, pp. 316-319, 2003.
[27] J.-S. Jang, "ANFIS: adaptive-network-based fuzzy inference system," *Systems, Man and Cybernetics, IEEE Transactions on*, vol. 23, pp. 665-685, 1993. Article (CrossRef Link)
[28] A. Shubair, S. Ramadass, and A. A. Altyeb, "kENFIS: kNN-based evolving neuro-fuzzy inference system for computer worms detection," *Journal of Intelligent and Fuzzy Systems*, 2014. Article (CrossRef Link)
[29] N. Mohdeb and M. R. Mekideche, "Determination of the relative magnetic permeability by using an adaptive neuro-fuzzy inference system and 2D-FEM," *Progress In Electromagnetics Research B*, vol. 22, pp. 237-255, 2010. Article (CrossRef Link)
[30] Android Malware Genome Project, 2014. http://www.malgenomeproject.org/
[31] Q. Liang and J. M. Mendel, "Interval type-2 fuzzy logic systems: theory and design," *Fuzzy Systems, IEEE Transactions on*, vol. 8, pp. 535-550, 2000. Article (CrossRef Link)
[32] A. Shubair and A. Al-Nassiri, "kEFCM: kNN-Based Dynamic Evolving Fuzzy Clustering Method," *International Journal of Advanced Computer Science and Applications (IJACSA)*, vol. 6 , no.2. Article (CrossRef Link)
[33] T. Fushiki, "Estimation of prediction error by using K-fold cross-validation," *Statistics and Computing*, vol. 21, pp. 137-146, 2011. Article (CrossRef Link)
[34] D. L. Olson and D. Delen, "Advanced data mining techniques," *Springer*, 2008.
[35] C. D. Katsis, N. Katertsidis, G. Ganiatsas, and D. I. Fotiadis, "Toward emotion recognition in car-racing drivers: A biosignal processing approach," *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, vol. 38, pp. 502-512, 2008. Article (CrossRef Link)
[36] R. Singh, A. Kainthola, and T. Singh, "Estimation of elastic constant of rocks using an ANFIS approach," *Applied Soft Computing*, vol. 12, pp. 40-45, 2012. Article (CrossRef Link)
[37] N. K. Kasabov and Q. Song, "DENFIS: dynamic evolving neural-fuzzy inference system and its application for time-series prediction," *Fuzzy Systems, IEEE Transactions on*, vol. 10, pp. 144-154, 2002. Article (CrossRef Link)
[38] C. Beleites, R. Salzer, and V. Sergo, "Validation of soft classification models using partial class memberships: An extended concept of sensitivity & co. applied to grading of astrocytoma tissues,"

Chemometrics and Intelligent Laboratory Systems, vol. 122, pp. 12-22, 2013. Article (CrossRef Link)

[39] T. Fawcett, "An introduction to ROC analysis," *Pattern recognition letters*, vol. 27, pp. 861-874, 2006. Article (CrossRef Link)



Shubair Abdulla received his BSc degree in computer science from Basra University in 1994. He received his Ms and PhD degrees in computer science from University Sains Malaysia (USM) in 2007 and 2014 respectively. Currently, he is working at Sultan Qaboos University, Oman, Muscat. His research interests include data mining, network security, and fuzzy inference systems.

Altyeb Altaher Altyeb received his B.Sc (Computer Science) in 2000 from the International University of Africa – Sudan, and an M.Sc. and Ph.D. (Computer Science) in 2002 and 2007, respectively from Khartoum University – Sudan. He worked as post-doctoral research fellow from 2010 to 2012 at the University Sains Malaysia (USM) – Malaysia. Currently he is an associate professor at King Abdulaziz University. His research interests lie in mobile and computer networks.