Intelligent Hybrid Approach for Android Malware Detection based on Permissions and API Calls

Altyeb Altaher, Omar Mohammed Barukab
Department of Information Technology, Faculty of Computing and Information Technology-Rabigh,
King Abdulaziz University, P.O. Box 344,
Jeddah, Saudi Arabia

***Abstract*—Android malware is rapidly becoming a potential threat to users. The number of Android malware is growing exponentially; they become significantly sophisticated and cause potential financial and information losses for users. Hence, there is a need for effective and efficient techniques to detect the Android malware applications. This paper proposes an intelligent hybrid approach for Android malware detection using the permissions and API calls in the Android application. The proposed approach consists of two steps. The first step involves finding the most significant permissions and Application Programming Interfaces (API) calls that leads to efficient discrimination between the malware and good ware applications. For this purpose, two features selection algorithms, Information Gain (IG) and Pearson CorrCoef (PC) are employed to rank the individual permissions and API's calls based on their importance for classification. In the second step, the proposed new hybrid approach for Android malware detection based on the combination of the Adaptive neural fuzzy Inference System (ANFIS) with the Particle Swarm Optimization (PSO), is employed to differentiate between the malware and goodware Android applications (apps). The PSO is intelligently utilized to optimize the ANFIS parameters by tuning its membership functions to generate reliable and more precise fuzzy rules for Android apps classification. Using a dataset consists of 250 goodware and 250 malware apps collected from different recourse, the conducted experiments show that the suggested method for Android malware detection is effective and achieved an accuracy of 89%.**

Keywords*—Android malware detection; features selection; fuzzy inference system; particle swarm optimization

I. INTRODUCTION

Recently, the use of smartphones in all aspects of our daily lives is increasing continuously. The global shipments of smartphones hit a record 1.4 billion in 2015 [1]. This number has grown 12% compared with the last year. The massive popularity of smartphones have been accompanied with a potential increase in the number of malwares. With Android dominating 82.8% of the market in 2015 [2], Android become the main goal for mobile malware. The number of Android malware applications is increasing continuously. The total number of malware attacking the mobile devices increased more than three times in 2015, compared to that of 2014 [3]. The dangerous threats targeting mobile devices in 2015 were ransomware. Malware can access all the resources in the attacked mobile device, and data stealers, like business malware.

Google's Play store is a market for Android apps, also there are many other third-party stores for Android apps. The Android apps developers use the Google's Play and third-party stores to publish the apps they developed, and make it available for download and install by users. Detecting the huge number of Android malware and isolating them from application markets is potential and great challenging issue. Very recently in 2016, a significantly sophisticated new form of Android ransomware/Android.Lockdroid.E is detected by Symantec, this variant of ransomware malware employs the accessibility tapjacking method to pose a real threat for more than 67% of Android devices [4].

Several research efforts have been presented for malware detection depending on the Android permissions used in the app. However, using Android permissions only is not enough for accurate detection of malware [5], [6]. Moreover, the existence of permissions in the Android application's Manifest.xml is not evident that it has been used by app code [7], [8]. On the other hand, some researches [9] consider the API level information only to get the features from big data set, but it requires a large number of features for the discrimination between malware and goodware apps. Moreover, efficient detection of the new and ever-evolving Android malware is a continues challenge. To address these challenges, this paper proposes a new hybrid method for Android malware detection based on the hybridization of the Adaptive neural fuzzy Inference System (ANFIS) with the Particle Swarm Optimization (PSO). This paper has the following contributions:

1) Finding the most significant permissions and API calls that lead to efficient categorization of malware and goodware apps.
2) Designing and implementing a new hybrid approach for Android malware detection based on the combination of (ANFIS) with (PSO).
3) An accurate dataset was collected which consists of 250 goodware and 250 malware apps from different resources including Google's play.

The rest of this paper is structured as: Section 2 presents the related work. Section 3 introduced the employed features and feature selection methods. Section 4 explains the proposed method for Android malware detection. Section 5 presents the experimental results and discussion. Section 6 includes the conclusion and future work.

II. Related Work

Android malware detection has been a very active research area in the recent years. The two main approaches for Android malware analysis are static and dynamic analysis [10]. Static analysis examines the Android application without executing it. Many static analysis approaches for Android malware detection have been proposed [11]-[15]. For example, Kirin's approach [12], explores the used permissions in the Android app to determine their maliciousness. Stowaway [13] examines API calls to detect Malware applications and Risk Ranker [14] statically classifies applications based on different security risks. In 2014, DREBIN proposed by Arp *et al*. [15], it detects the Android malware application by performing application analysis using different features such as permissions, API calls, network address, hardware access, etc. Examples for static analysis tools include Smali [16] and Androguard [17], which facilitate the static analysis of Android apps. However, the static analysis approaches are unable to detect malwares that use dynamic code loading obfuscation techniques.

The dynamic analysis examines the Android application by monitoring its behavior during execution. A number of dynamic analysis approaches have been proposed for Android malware detection [18]-[22]. DroidScope and TaintDroid proposed in [21] and [22], they monitor the application during its running in a protected environment (similar to the java sand box concept), by exploring different components of the application. Dynamic analysis approaches require many resources compared with static analysis, which made them unsuitable for the limited resources of the mobile devices.

Machine learning approaches also have been proposed for Android malware detection, motivated by the problem of manually creating and updating detection models for Android malware. Aafer *et al*. [23] used machine learning approach to classify Android apps as malware or goodware. They compared the malware detection accuracy of four classifiers using the API calls and permissions as features for Android application. Shabtai *et al*. [24] used six machine learning based classifiers for Android malware detection, namely, k-Means, NB, DT, decision tree, BN, logistic regression and histogram using the Andromaly framework. Andromaly achieved a 99.9% accuracy rate using the information gain algorithm for features selection and decision tree classifier. The main disadvantage of Andromaly is using self-written Android malware applications to test it. In contrast, 250 real-world malware apps were used in our approach.

Dini *et al*. in [19] combined the system calls in kernel with system calls in user level. They used features set consisting of 12 system calls and the K-nearest neighbors (KNN) as classifier for Android malware detection. Dini *et al*. achieved an accuracy of 93% using 10 Android malwares. The main disadvantage of this approach is its inability to detect malware that use root permissions to avoid the system call. Therefore, Android API calls and permissions are used in this research rather than using system calls only. Zhao *et al*. in [25] used the Support Vector machine learning method to develop RobotDroid in order to detect unknown Android malware. RobotDroid depends on the hidden payment services and the privacy information leakage. They considered three malware types, namely, Plankton, DroidDream and Gemini. Consequently, this approach is limited to the considered Android malware types only. Amos *et al*. proposed STREAM in [26], it employs several machine learning classifiers to detect Android malware based on a set of features such as permissions , memory and battery usage. However, the features set collected from Android emulator, which is not accurate as real Android device [27]. In this work, we used features collected from real-world malware and goodware apps.

Our proposed approach is related to these methods and uses comparable features for detecting malicious applications, such API calls and permissions. However, it differs in two main aspects from previous research: First, we find the most significant permissions and API calls that leads to efficient discrimination between the malware and goodware applications. Second, we design and implement a new hybrid classifier for efficient Android malware detection based on combining (ANFIS) [35] with the PSO.

III. Overview

A. Android Permissions

Android uses the permission system to regulate the application's access to the resources and data in the mobile device, such as camera, storage and internet access. Android permissions offer the security characteristics by restricting certain tasks an application can execute [28]. The selection of the most suitable permissions corresponding to the tasks performed by the application is the ultimate responsibility of the developer. The shortage of knowledge about the permissions could guide developers to add in essential and excessive permissions to the application which may leads to over privileged Android application [29] that make users stop installing the application. Moreover, adding in essential permissions makes the application similar to malware [30], causing re-delegation attacks [31]. Furthermore, the shortage of knowledge about permissions makes the user uncertain about the decision of installing the application. Currently available information about permissions is not quite enough to support the users decision regarding the installation of application [32]. Since the used permissions and API calls in the application reflect the behavior of the app, it is believed that it is potentially feasible to detect the malware application by exploring the patterns of used permissions and API calls, and help both developers and users in getting better understanding of permissions and API calls.

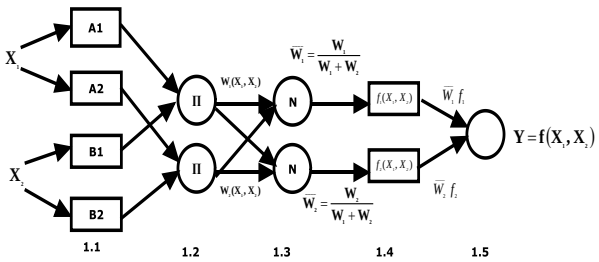

Fig. 1. Artificial neuro fuzzy inference system architecture.

B. Android API Calls

APIs are functions used by the developers to interact with Android operating system. Since there are many Android APIs, it is more appropriate to consider specific APIs frequently used by malware and enable them to access Smartphone's sensitive resources, rather than considering all APIs used in the app's source code. Seo *et al*. [33] examined malware apps and specified suspicious APIs frequently used by malware. They compared the number of suspicious APIs in malware applications with the number of suspicious APIs in goodware apps. In this paper, the suspicious APIs from the malware and goodware apps were extracted and used as part of the features set. The selected suspicious APIs are similar to the subspecies APIs defined in [33], [15], examples of the selected suspicious APIs include, API calls for accessing important information such as getSubscriberId()and getDeviceId(), and API calls for getting accessibility services which can activate several types of malicious payloads. Finding the rank of the importance of API calls and permissions in terms of categorization of malware and goodware application, and evaluating the efficiency of our proposed hybrid approach PSO-ANFIS for malware detection, are main challenges of this research work.

C. Artificial Neuro Fuzzy Inference System (ANFIS)

Artificial Neuro Fuzzy Inference System (ANFIS) is proposed by Jang *et al*. [34] and it became one of the well-known researches related to fuzzy inference systems in the recent years. ANFIS associates the advanced reasoning and explanation of fuzzy logic with the powerful computation and learning capabilities of neural networks [34]. There are many architectures for neuro-fuzzy networks, however, the most dominant architectural models are the Mamdani and the Takagi Sugeno (TSK). TSK model uses a single-spike as the output membership function while Mamdani model uses a fuzzy set. The ANFIS used in this paper is based on TSK model. Fig. 1 shows example for ANFIS architecture that consists of five layers with two inputs X1 and X2 and one output Y.

The rule base consists of Takagi–Sugeno fuzzy rules as follows:

R1=IF x1 is μ1i AND x2 is μ2i THEN fij = pijx1 + qijx2 + rij

Where, x1 and x2 are the input for the ANFIS, conditions specified in the IF part are called antecedent, and conditions in the THEN part are called the consequence. ANFIS consists of five layers [34].

Layer One: Crisp input data entered into the ANFIS are transformed into linguistic expressions. Membership function (μ) is used to transform the inputs into linguistic expressions, there are several types of membership functions. In this paper, Gaussian membership function is used because it leads to more smooth model behavior. The Gaussian membership function is defined as follow:

$$O_i^1 = \beta(X) = \exp\left(-\frac{1}{2}\frac{(X-C)^2}{\sigma^2}\right) \quad (1)$$

Where, Oi's are the outputs of this layer. σ and C represent the variance and the center of the Gaussian membership function, respectively. This layer includes parameters called antecedent parameters, which are tuned by ANFIS for obtaining results that are more accurate.

Layer 2: This layer determines the level of accuracy of the statements specified in antecedent parts, which is also called the firing strength, using the following equation.

$$O_i^3 = \frac{w_i}{\sum_i w_i} \quad (2)$$

Layer 3: This layer normalizes all the firing strengths computed in the former layer (Wi), where it computes the proportion of the ith rule's firing strength to the all rules firing strengths using the following equation:

$$O_i^3 = \frac{w_i}{\sum_i w_i} \quad (3)$$

Layer 4: This layer contains the consequent portion of the fuzzy rules, the influence of each rule in the final output is defined as:

$$O_i^4 = \overline{W_i} f_i = \overline{W_i}(m_i X_1 + n_i X_2 + r_i) \quad (4)$$

Where, mi, ni, and ri are the consequent parameters. The consequent parameters and the antecedent parameters in the first layer, are tuned by ANFIS in the learning process to decrease the differences between the output and the target result.

Layer 5: This is the defuzzification layer where all the rules generated for one output are collected and defuzzified into numerical outputs, according to a weighted average sum as follow:

$$O_i^5 = Y = \sum_i \overline{W_i} f_i = \overline{W_1} f_1 + \overline{W_2} f_2 = \frac{\sum_i W_i f_i}{\sum_i W_i} \quad (5)$$

D. The Particle Swarm Optimization (PSO)

The particle swarm optimization (PSO) is a well-known method for searching, which is grounded on the behavior of bird flocking. In PSO, each possible answer for the optimization issue can be considered as a point in the population, the point is called a particle. The particle propagates in the population with a specified speed which is tuned based on its movement knowledge and its companions' movement knowledge. Each particle is assigned a fitness value specified based on the objective function and records its current position and current best position (recorded as pbest). The pbest can be considered as the particle's own moving experience. Moreover, each particle records the global best position (recorded as gbest, which is the best value in pbest) of the total collection. The gbest can be considered as its companions' movement knowledge for the particle [35], [36]. Let xi (t) represent the location of a particle i in the population at time t. Then, the particle's location is adjusted by the addition of the velocity, vi to the current location:

$$x_i(t) = x_i(t) + v_i(t) \quad (6)$$

$$v_i(t) = c_1 r_1\big(pbest(t) - x_i(t)\big) + c_2 r_2\big(gbest(t) - x_i(t)\big) \quad (7)$$

Where, c1 and c2 represent the acceleration coefficients, r1 and r2 are random vectors.

IV. The Proposed Approach for Android Malware Detection

This section explains our methodology for the detection of Android malware. First, the used dataset is introduced. Second, we used the Information Gain (IG) and Pearson CorrCoef (PC) to find the most significant permissions and API calls that leads to efficient discrimination between the malware and goodware applications. Third, the proposed hybrid PSO-ANFIS is utilized to classify the Android apps as either goodware or malware.

A. The Used Dataset

To build an accurate and reliable model capable of efficiently classifying the Android applications as malware or goodware, an accurate and well-labeled dataset is required. We used a recent data set contains 250 benign applications collected from Google's play, the free online malware detection tool VirusTotal [36] is used to confirm that the collected apps are benign. The data set also contains 250 malware applications which have been downloaded from Genome project [37] and Drebin dataset [15], all the malware apps are confirmed as malware using the VirusTotal. In this paper, we considered the permissions and API calls used in the app as features to classify the applications as either goodware or malware.

B. Features selection

In this section, the most significant permissions and API calls that leads to efficient discrimination between the malware and goodware applications are selected. For this purpose, two features ranking algorithms, namely, Information Gain (IG) and Pearson CorrCoef (PC) are employed to rank the individual permissions and API calls based on their importance for classification. Based on the static analysis method, we used free analysis tools like android-apktool, dex2jar and jd-gui to extract the API calls and permissions from the goodware and malware applications. Java language is used to develop Android applications and they are generated as Android Application Package (APK) files. APK package consists of files required for running the application. The files in the APK include the following:

DalVik Executable (DEX) file: This is a file generated by the compilation of the Java source code.

Manifest file: A file holding the Android application characteristics such as permissions, the activities, and intents.

eXtensible Markup Language (XML) file: A file defining the components of the user interface layout and the used values.

Resource file: A file consists of resources needed by applications like sound files and images.

- *Information Gain Ratio (IGR)*

Information Gain Ratio method extracts the correlation between Android API calls and permissions, it gives the maximum scoret to the most potential permissions based on the class of malware and goodware Android apps belonging to IGR [38], the IGR equations are explained as follows:

$$gain\_r(X, C) = \frac{gain\_r(X,C)}{split\_info(C)} \quad (8)$$

$$split\_info(C) = \sum_i \left(\frac{|C_i|}{|C|}\right) log \frac{|C_i|}{|C|} \quad (9)$$

Where, gain_r (X,C) denotes the gain ratio of the feature X occurrence in the class C. Ci and |Ci| represent the occurrence of feature X in class C, the ith sub-class of C and the total number of features in Ci, respectively.

- *The Pearson CorrCoef*

Pearson CorrCoef computes the relation between feature X and class C by

$$R(X, C) = \frac{cov(X,C)}{\sqrt{var(X)var(C)}} \quad (10)$$

Where, R(P;A) = 0 means the independency of feature P and class A, R(P;A) = 1 means the top positive correlation of feature P and class A and R(P;A) = -1 means the top negative correlation. In our paper, R(P;A) = 1 means that the application which request the feature P is highly suspected as malware app, while R(P;A) = -1 means requesting the feature P leads to classify the applications as goodware.

C. PSO-based ANFIS

The proposed approach PSO-ANFIS intelligently combines ANFIS method and PSO algorithm for the optimization of Android malware detection, by tuning the parameters of membership function to achieve the highest malware detection accuracy. ANFIS utilizes the IF-Then fuzzy rules in mapping the inputs to outputs, the level of ANFIS accuracy is significantly affected by tuning its network structure and parameters [40]. There is a strong relation between the accuracy and the used membership functions. This paper aims to tune the Gaussian membership function to improve the accuracy of ANFIS by getting the best learning parameters and minimum number of fuzzy rules. The best learning parameters which minimize the difference between the target data and ANFIS output, can be obtained by minimizing the Root Mean Square Error (RMSE) as follows:

$$J = \left[\sum_{m=1}^{\alpha} \frac{(R_m - y_m)^2}{\alpha}\right]^{\frac{1}{2}} \quad (11)$$

Where, α represents the number of training data items, Rm is the input value and ym is the predicted value. The Performance criterion RMSE depends on the center and the width of the Gaussian membership function (Ci, σi). The RMSE is considered as an objective function and its minimization is the research problem in this study. PSO is a potential optimization technique to enhance the performance of ANFIS [41]. In this paper, PSO algorithm is utilized intelligently to realize the minimization of the objective function of ANFIS which is the error between the predicted class of Android malware and the actual class, by tuning the membership functions of ANFIS. Fig. 2 shows the structure of the proposed hybrid PSO-ANFIS approach, each learning process of ANFIS represents one particle and the parameters of the membership functions that effect the ANFIS performance represent the particle dimensions.

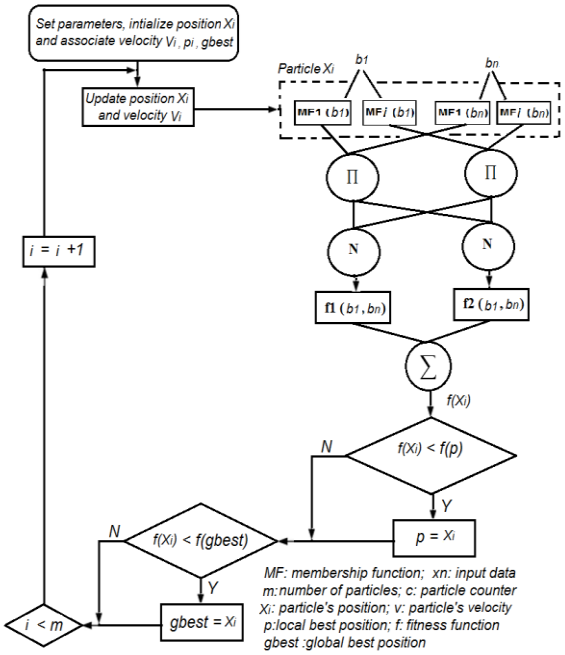

Fig. 2. The proposed POS-ANFIS classifier.

The steps of the hybrid PSO-ANFIS approach are explained as follows:

Step 1: Set initial value for the number of iterations C =1, randomly initialize the position Xiand velocity Viof the i-th particle in the search space using (6) and (7), respectively.

Step 2: Compute the objective function of ANFIS using (8) and use it as a fitness function (f_i) for the proposed PSO-ANFIS.

Step 3: IF the fitness function of the i-th particle in position X $f(X_i)$ is better than the fitness function of the same particle in the local best position f (Pi) THEN

$P_i = X_i$.
ELSE
IF the fitness function of the i-th particle in position X $f(X_i)$ is better than the fitness function of the same particle in the global best position$f(gbest)$ THEN
Gbest = Xi

Step 4: IF the fitness function of global best position$f(gbest)$ is best than stopping criteria OR the particle counter C is greater than the total number of iterations m THEN

Stop
ELSE
Go to Step 2

V. Experiments and Results Discussion

To conduct the experiments the dataset is separated into two sets testing set and training set. The five-fold cross-validation technique is selected to split the dataset. In order to evaluate the performance of the proposed approach for Android malware detection, we selected the 24 permissions and API calls features according to their significance for the discrimination, using the Pearson CorrCoef (PC) and Information Gain (IG) algorithms. We applied our proposed approach PSO-ANFIS on the selected features. The performance of our proposed approach PSO-ANFIS is compared with the well-known system ANFIS system. ANFIS is selected for comparing our proposed PSO-ANFIS with, because it proofs efficiency in many classification tasks and it is similar to PSO-ANFIS in terms of fuzzy rules type. The classification accuracy of the two classifiers is compared in terms of the Root Mean Square Error (RMSE). Root Mean Square Error measures the dissimilarity between values obtained by the classifier and the observed values:

$$RMSE = \sqrt{\frac{1}{2}\sum_{i=1}^{n}(M_i - P_i)^2} \qquad (12)$$

The total accuracy obtained by the cross-validation is calculated based on the average of the RMSE in the five-folds method as follows:

$$CVA = \frac{1}{m}\sum_{i=1}^{m} R_i \qquad (13)$$

Where, CVA is the accuracy based on the cross-validation, m is the number of the used folds, and Ri is the calculated RMSE of each fold. The evaluation of the fuzzy inference classifier requires the separation of its output into two main classes. To evaluate our proposed POS-ANFIS, we separated its continuous output by determining a threshold in the range between one and zero, to categorize the malware and goodware apps. In this experiment, four threshold values have been determined: 0.10, 0.25, 0.35 and 0.40. The classification results based on the selected features are compared in terms of Accuracy (ACC) and obtained error rate. The following confusion matrix used to classify the Android application as malware (1) or goodware (0).

		Actual	
		1	0
Classified	1	TP	FP
	0	FN	TN

Where,

TP: correct classification of malware as malware

FP: incorrect classification of malware as goodware

FN: incorrect classification of goodware as malware

TN: correct classification of goodware as goodware.

TABLE. I. MOST FREQUENTLY USED FEATURES IN MALWARE AND GOODWARE APPLICATIONS AND THEIR RANK OF IMPORTANCE

Information Gain Algorithm			**Pearson CorrCoef Algorithm**		
Rank	***Score***	***Feature***	***Rank***	***Score***	***Feature***
1	0.2746	READ_PHONE_STATE	1	0.5811	READ_PHONE_STATE
2	0.2743	getDescription	2	0.5563	RECEIVE_BOOT_COMPLETED
3	0.2537	AccessibilityNodeProvider	3	0.5542	SYSTEM_ALERT_WINDOW
4	0.2537	AccessibilityNodeInfo	4	0.5403	getDescription
5	0.2537	AccessibilityRecord	5	0.5186	AccessibilityRecord
6	0.2537	AccessibilityStateChangeListener	6	0.5186	AccessibilityNodeProvider
7	0.2537	AccessibilityManager	7	0.5186	AccessibilityNodeInfo
8	0.2529	SYSTEM_ALERT_WINDOW	8	0.5186	AccessibilityManager
9	0.2366	RECEIVE_BOOT_COMPLETED	9	0.5186	AccessibilityStateChangeListener
10	0.0816	SEND_SMS	10	0.3151	SEND_SMS
11	0.0776	INTERNET	11	0.3115	INTERNET
12	0.0473	ACCESS_NETWORK_STATE	12	0.2544	ACCESS The STATE Of NETWORK
13	0.0414	WRITE_EXTERNAL_STORAGE	13	0.2378	WRITE In The EXTERNAL STORAGE
14	0.0124	READ_SMS	14	0.1283	READ_SMS
15	0.0112	WRITE_SETTINGS	15	0.0427	WRITE_SETTINGS
16	0.0109	getResolveInfo	16	0.0161	getResolveInfo
17	0.0101	KILL_BACKGROUND_PROCESSES	17	0.0104	KILL_BACKGROUND_PROCESSES
18	0.0019	WAKE_LOCK	18	0.0041	WAKE_LOCK
19	0.0012	getSettingsActivityName	19	0.0040	getSettingsActivityName
20	0.0010	GET_TASKS	20	0.0004	GET_TASKS

TABLE. II. COMPARISON OF TESTING AND TRAINING RMSE FOR BOTH ANFIS AND PSO-ANFIS

Fold #	**ANFIS**		**PSO-ANFIS**	
	Testing RMSE	**Training RMSE**	**Testing_RMSE**	**Training RMSE**
1	0.3318	0.3476	0.1574	0.1168
2	0.3221	0.3296	0.1161	0.1053
3	0.3159	0.3282	0.1458	0.1245
4	0.3147	0.3156	0.1494	0.1265
5	0.3113	0.3396	0.1359	0.1184

Table 1 above shows the most frequently used features in malware and goodware applications and their rank of importance. It is observed that using the Information gain and Pearson CorrCoef Algorithm yield the same subset of 15 features, signifying that the results of ranking are consistent. The most requested permissions are associated with accessing phone state, connecting to the Internet, checking the connectivity of the network, monitoring the device booting, writing to external storage and messages related permissions. Moreover, most of the applications requested the SYSTEM_ALERT_WINDOW permission and accessibility services APIs. They enable an application to create windows that can be shown on top of other applications and even execute tasks on them, these abilities made them an extremely attractive target for malware developers to perform tapjacking attacks. A significantly sophisticated new form of Android ransomware/Android.Lockdroid.E is detected [4], this variant of ransomware malware employs the accessibility tapjacking method to pose a real threat for more than 67% of Android devices.

Table 2 above shows the testing and training RMSE for both ANFIS and PSO-ANFIS using the features selected by the proposed dual-stage method based on the five-fold cross-validation. A low RMSE depicts greater malware detection accuracy, so the classifier that generates lower RMSE is considered as a better classifier for the differentiation between the malware and goodware applications. It can be observed from Table 2 that the proposed PSO-ANFIS achieves lowest RMSE for both testing and training in all five-folds.

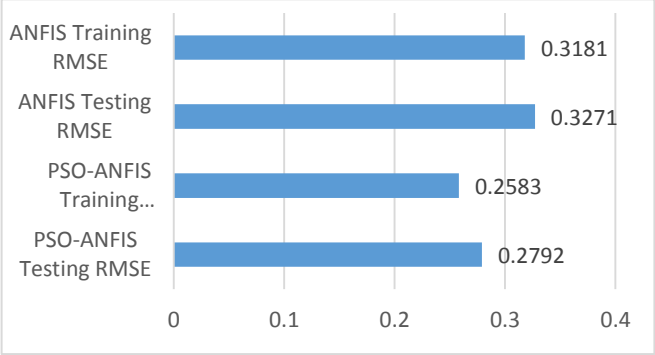

Fig. 3. The accuracy of cross-validation in terms of RMSE.

TABLE. III. CLASSIFICATION ACCURACY OF PSO-ANFIS AND ANFIS

Threshold	**Accuracy**		**Error**	
	PSO-ANFIS	**ANFIS**	**PSO-ANFIS**	**ANFIS**
0.10	45 %	41%	55 %	59%
0.25	63 %	50%	37 %	50%
0.35	66 %	54%	34 %	46%
0.40	89%	69%	11 %	31%

TABLE. IV. THE PERFORMANCE OF THE PROPOSED APPROACH COMPARED WITH OTHER APPROACHES

Classification approach	**Accuracy**
PSO-ANFIS	89%
K-ANFIS [39]	75%
Droid Permission Miner [42]	87%
Puma [5]	86.41%.

Table 3 above shows the performance of PSO-ANFIS and ANFIS for each threshold in terms of accuracy and error rates. We can see that the proposed classifier PSO-ANFIS achieved better results than ANFIS for all thresholds. These results confirm the suitability of PSO algorithm in minimizing the error between the predicted class of Android application and the actual class, by tuning the membership functions of ANFIS. Fig. 3 shows a comparison between the values of CVA for PSO-ANFIS and ANFIS. It is clear from Fig. 3 that the results from PSO-ANFIS are significantly better. For example, the CVA of RMSE value of PSO-ANFIS testing is 0.2792 which is smaller than the CVA value of ANFIS testing which is 0.3271. Also, the CVA of RMSE value of PSO-ANFIS training is 0.2583 which is smaller than the CVA value of ANFIS training which is 0.3181. All these results confirm that PSO-ANFIS system outperforms ANFIS. The performance of our proposed approach has been compared with other comparable approaches, including our previous research [39] and other approaches [5], [42] as shown in Table 4.

VI. Conclusion

Accurate detection of Android malware has been an important issue in recent years. In this study, we found that the most significant permissions and API calls lead to efficient discrimination between the malware and goodware applications. For this purpose, two features ranking algorithms, Information Gain (IG) and Pearson CorrCoef (PC) are employed to rank the individual permissions and API calls based on their importance for classification. In addition, we proposed a new hybrid method for Android malware detection based on the combination of the Adaptive neural fuzzy Inference System (ANFIS) with the Particle Swarm Optimization (PSO). Using dataset consists of 250 malware and 250 goodware collected from different recourse, the proposed approach achieved classification accuracy of 89% which is better than the classification accuracy of ANIS and other approaches. For future work ensemble of classifiers will be considered to improve the classification accuracy of Android apps.

Acknowledgement

This work was supported by the Deanship of Scientific Research (DSR), King Abdulaziz University, Jeddah, under Grant No. (G-160-830-37). The authors, therefore, gratefully acknowledge the DSR technical and financial support.

References

[1] Rawal, Priyanka, Alka Awasthi, and Shekhar Upadhayay. “Creating a Hunger Driven Smartphone Market by Xiaomi.” International Journal of Engineering Science, pp. 11250-11255,2017.

[2] Azfar, Abdullah, Kim-Kwang Raymond Choo, and Lin Liu. “Forensic taxonomy of android productivity apps.” Multimedia Tools and Applications .76, no. 3 ,pp. 3313-3341, 2017.

[3] Kambourakis, Georgios, Dimitrios Damopoulos, Dimitrios Papamartzivanos, and Emmanouil Pavlidakis. “Introducing touchstroke: keystroke‐based authentication system for smartphones.” Security and Communication Networks, 9, no. 6, pp. 542-554, 2016.

[4] Narudin, Fairuz Amalina, Ali Feizollah, Nor Badrul Anuar, and Abdullah Gani. “Evaluation of machine learning classifiers for mobile malware detection.” Soft Computing, 20, no. 1, pp.343-357, 2016

[5] Sanz, B., Santos, I., Laorden, C., Ugarte-Pedrero, X., Bringas, P. G., & Álvarez, G. Puma: Permission usage to detect malware in android. InInternational Joint Conference CISIS’12-ICEUTE´ 12-SOCO´ 12 Special Sessions, Springer Berlin Heidelberg, pp: 289-298, 2013.

[6] Wei, X., Gomez, L., Neamtiu, I., & Faloutsos, M. Permission evolution in the android ecosystem. In Proceedings of the 28th Annual Computer Security Applications Conference, ACM, pp: 31-40,2012.

[7] Sharma, Akanksha, and Subrat Kumar Dash. Mining API Calls and Permissions for Android Malware Detection. In *Cryptology and Network Security*, Springer International Publishing, pp: 191-205, 2014.

[8] Grace, M. C., Zhou, Y., Wang, Z., & Jiang, X. Systematic Detection of Capability Leaks in Stock Android Smartphones. In NDSS ,Vol. 14,. pp. 19,2012.

[9] Aafer, Y., Du, W., & Yin, H. Droidapiminer: Mining api-level features for robust malware detection in android. In International Conference on Security and Privacy in Communication Systems ,Springer International Publishing, pp: 86-103,2013.

[10] Egele, Manuel, Theodoor Scholte, Engin Kirda, and Christopher Kruegel. “A survey on automated dynamic malware-analysis techniques and tools.” ACM Computing Surveys (CSUR) 44, no. 2,pp. 6, 2012.

[11] Enck, W., Octeau, D., McDaniel, P., & Chaudhuri, S. A Study of Android Application Security. In USENIX security symposium,Vol. 2 ,pp. 2. 2011.

[12] Enck, W., Ongtang, M., & McDaniel, P. On lightweight mobile phone application certification. In Proceedings of the 16th ACM conference on Computer and communications security, ACM, pp: 235-245, 2009.

[13] Felt, A. P., Chin, E., Hanna, S., Song, D., & Wagner, D. Android permissions demystified. In Proceedings of the 18th ACM conference on Computer and communications security ,ACM, pp.627-638,2011.

[14] Grace, M., Zhou, Y., Zhang, Q., Zou, S., & Jiang, X. Riskranker: scalable and accurate zero-day android malware detection. In Proceedings of the 10th international conference on Mobile systems, applications, and services, ACM, pp. 281-294, 2012.

[15] Arp, D., Spreitzenbarth, M., Hubner, M., Gascon, H., Rieck, K., & Siemens, C. E. R. T. DREBIN: Effective and Explainable Detection of Android Malware in Your Pocket. In NDSS, 2014.

[16] J. Freke. An assembler/disassembler for android’s dex format. Google Code, http://code.google.com/p/smali/, visited February, 2013.
Desnos, A., & Gueguen, G. Android: From reversing to decompilation. Proc. of Black Hat Abu Dhabi, pp.77-101, 2011

[17] Burguera, I., Zurutuza, U., & Nadjm-Tehrani, S. Crowdroid: behavior-based malware detection system for android. In Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices ACM, pp. 15-26,2011.

[18] Dini, G., Martinelli, F., Saracino, A., & Sgandurra, D. MADAM: a multi-level anomaly detector for android malware. In International Conference on Mathematical Methods, Models, and Architectures for Computer Network Security ,Springer Berlin Heidelberg,pp. 240-253, 2012.

[19] Zhou, Y., Wang, Z., Zhou, W., & Jiang, X. February. Hey, you, get off of my market: detecting malicious apps in official and alternative android markets. In NDSS ,Vol. 25, No. 4, pp. 50-52,2012.

[20] Yan, L. K., & Yin, H. DroidScope: Seamlessly Reconstructing the OS and Dalvik Semantic Views for Dynamic Android Malware Analysis. In USENIX security symposium, pp. 569-584,2012.

[21] Enck, W., Gilbert, P., Han, S., Tendulkar, V., Chun, B. G., Cox, L. P., ... & Sheth, A. N. TaintDroid: an information-flow tracking system for realtime privacy monitoring on smartphones. ACM Transactions on Computer Systems (TOCS), 32(2), 2014.

[22] Aafer, Y., Du, W., & Yin, H. Droidapiminer: Mining api-level features for robust malware detection in android. In International Conference on Security and Privacy in Communication Systems, Springer International Publishing, pp. 86-103, 2013.

[23]] Shabtai, A., Kanonov, U., Elovici, Y., Glezer, C., & Weiss, Y. “Andromaly”: a behavioral malware detection framework for android devices. Journal of Intelligent Information Systems, 38(1), pp. 161-190, 2012.

[24] Zhao, M., Zhang, T., Ge, F., & Yuan, Z. RobotDroid: A Lightweight Malware Detection Framework On Smartphones. JNW, 7(4), pp. 715-722,2012.

[25] Amos B, Turner H, White J. Applying machine learning classifiers to dynamic android malware detection at scale. In: Proceedings of the 9th international wireless communications and mobile computing conference (IWCMC), Sardinia, Italy, pp. 1666–1671, 2013.

[26] Raffetseder T, Kruegel C, Kirda E. Detecting system emulators.In: Proceedings of the 10th international conference ISC, Valparaíso, Chile, pp. 1–18,2007.

[27] Android, "Andriod developers guides,"

[28] http://developer.android.com/guide/topics/security

[29] Felt, A. P., Chin, E., Hanna, S., Song, D., & Wagner, D. Android permissions demystified. In Proceedings of the 18th ACM conference on Computer and communications security ,ACM, pp. 627-638, 2011.

[30] Grace, M. C., Zhou, Y., Wang, Z., & Jiang, X. Systematic Detection of Capability Leaks in Stock Android Smartphones. In NDSS ,Vol. 14, pp. 19,2012.

[31] Felt, A. P., Wang, H. J., Moshchuk, A., Hanna, S., & Chin, E. Permission Re-Delegation: Attacks and Defenses. In USENIX Security Symposium , 2011.

[32] Kelley, P. G., Consolvo, S., Cranor, L. F., Jung, J., Sadeh, N., & Wetherall, D. A conundrum of permissions: installing applications on an android smartphone. In Financial Cryptography and Data Security ,Springer Berlin Heidelberg, pp. 68-79,2012.

[33] Seo, S. H., Gupta, A., Sallam, A. M., Bertino, E., & Yim, K. Detecting mobile malware threats to homeland security through static analysis. Journal of Network and Computer Applications, 38, pp. 43-53,2014.

[34] Jang, J. S. ANFIS: adaptive-network-based fuzzy inference system. IEEE transactions on systems, man, and cybernetics, 23(3), pp. 665-685, 1993

[35] Trelea, I. C. The particle swarm optimization algorithm: convergence analysis and parameter selection. Information processing letters, 85(6), pp. 317-325, 2003.

[36] Kennedy, J., & Eberhart, R. Particle swarm optimization, IEEE International of first Conference on Neural Networks, 1995.

[37] Jarabek, Chris, David Barrera, and John Aycock. "ThinAV: truly lightweight mobile cloud-based anti-malware." In Proceedings of the 28th Annual Computer Security Applications Conference, pp. 209-218, 2012.

[38] Zhou, Y. and Jiang, X. Dissecting android malware: Characterization and evolution. In Security and Privacy (SP), 2012 IEEE Symposium on ,pp. 95-109, 2012.

[39] Mori, T. Information gain ratio as term weight: The case of summarization of IR results. Proceedings of the 19th International Conference on Computational Linguistics, (COLING '02), Association for Computational Linguistics, pp. 1-7,2002.

[40] Abdulla, S., & Altaher, A. Intelligent Approach for Android Malware Detection. KSII Transactions on Internet and Information Systems (TIIS), 9(8),pp. 2964-2983, 2015.

[41] Jiang, H.M., Kwong, C.K., Ip, W.H. and Wong, T.C. Modeling customer satisfaction for new product development using a PSO-based ANFIS approach. Applied Soft Computing, 12(2), , pp.726-734,2012.

[42] T.Fushiki. Estimation of prediction error by using K-fold cross-validation, Statistics and Computing,vol.21, pp.137-146,2011.

[43] Aswini, A.M., Vinod, P. Droid Permission Miner: Mining Prominent Permissions for Android Malware Analysis. In: 5th International Conference on the Applications of the Digital Information and Web Technologies (ICADIWT 2014), pp. 81–86 ,2014.