Neural Comput & Applic (2017) 28:4147–4157
DOI 10.1007/s00521-016-2708-7

ORIGINAL ARTICLE

An improved Android malware detection scheme based on an evolving hybrid neuro-fuzzy classifier (EHNFC) and permission-based features

Altyeb Altaher[1]

Received: 17 September 2016 / Accepted: 4 November 2016 / Published online: 16 November 2016

Abstract The increasing number of Android devices and users has been attracting the attention of different types of attackers. Malware authors create new versions of malware from previous ones by implementing code obfuscation techniques. Obfuscated malware is potentially contributed to the exponential increase in the number of generated malware variants. Detection of obfuscated malware is a continuous challenge because it can easily evade the signature-based malware detectors, and behaviour-based detectors are not able to detect them accurately. Therefore, an efficient technique for obfuscated malware detection in Android-based smartphones is needed. In the literature on Android malware classification, few malware detection approaches are designed with the capability of detecting obfuscated malware. However, these malware detection approaches were not equipped with the capacity to improve their performance by learning and evolving their malware detection rules. Based on the concept of evolving soft computing systems, this paper proposes an evolving hybrid neuro-fuzzy classifier (EHNFC) for Android malware classification using permission-based features. The proposed EHNFC not only has the capability of detecting obfuscated malware using fuzzy rules, but can also evolve its structure by learning new malware detection fuzzy rules to improve its detection accuracy when used in detection of more malware applications. To this end, an evolving clustering method for adapting and evolving malware detection fuzzy rules was modified to incorporate an adaptive procedure for updating the radii and centres of clustered permission-based features. This modification to the evolving clustering method enhances cluster convergence and generates rules that are better tailored to the input data, hence improving the classification accuracy of the proposed EHNFC. The experimental results for the proposed EHNFC show that the proposal outperforms several state-of-the-art obfuscated malware classification approaches in terms of false negative rate (0.05) and false positive rate (0.05). The results also demonstrate that the proposal detects the Android malware better than other neuro-fuzzy systems (viz., the adaptive neuro-fuzzy inference system and the dynamic evolving neuro-fuzzy system) in terms of accuracy (90%).

Keywords Android security · Malware · Malware detection · Evolving clustering algorithm · Evolving hybrid neuro-fuzzy classifier

1 Introduction

Recently, the number of mobile phones that use Android as an operating system has been increasing rapidly [1]. According to Gartner's research [2], Android is on pace to exceed one billion users across all devices. Google Play is the official market for both free and paid smartphone applications (hereafter 'apps') [3]. Google Play contains approximately one million apps, grouped into different classifications, such as business, books, games and security. In order to upload apps to Google Play, developers must first create an account, after which Google Play uses a service called Bouncer to detect malicious apps. Bouncer uses dynamic and static analysis to determine the behaviour of apps [4]. Android-based devices have become a

✉ Altyeb Altaher
altypaltaher@gmail.com

[1] Faculty of Computing and Information Technology in Rabigh, King Abdulaziz University, Rabigh 21911, Saudi Arabia

potential target for attackers to obtain confidential information and steal money. A recent report about the IT Threat Evolution for Q1 of 2015 by Kaspersky reports that 103,072 new malicious mobile applications were detected in the first quarter of 2015, and the number of new malicious mobile apps detected in the first quarter of 2015 is 3.3 times the number of new malicious mobile applications detected in the final quarter of last year. Kaspersky reported that mobile malware developed to steal user's money like banker Trojans and SMS Trojans accounted for 23.2% of new mobile malware in the first quarter of 2015. Money-stealing malware is really dangerous, and the malware authors' concentration in their victims' money provides a motivation for their further development and evolution [5, 6]. Also the malware authors continuously create new versions of malware from previous ones by implementing code obfuscation techniques, which can automatically convert an original version of malware to many other new variants using different techniques such as changing the names of the methods or the order of its execution. Thus, to avoid malware detectors, malwares evolve their code into updated generations using the obfuscation technique. Malware obfuscation is potentially contributed to the exponential growth in the number of generated malware variants [7].

Considering the challenges of detecting obfuscated malware, I propose an evolving hybrid neuro-fuzzy classifier (EHNFC) based on an evolving intelligent system to detect obfuscated malware. The proposal in this research is different from the other previous obfuscated malware detectors in that it addresses two limitations. First, signature-based detectors face the challenge of detecting obfuscated malware, as they depend on signatures to identify malware. However, signatures of obfuscated malware are changing frequently and can only be available if samples are available. It is continuously a challenge for signature-based classifiers to classify the obfuscated malware and update the malware signature database as fast as possible. EHNFC addresses this issue by dynamically creating and evolving high-quality fuzzy rules to perfectly predict the newly generated malware based on mining the previous fuzzy rules. Second, behaviour-based detectors consume the limited resources (power, processing and memory) of the smartphone and they are not able to detect the obfuscated malware accurately. To address the issue of the limited resources of smartphones, the proposed EHNFC uses a suitable feature selection method at the preparatory stage, to detect the malware efficiently with a faster classification process and a lower classification time.

EHNFC learns to classify Android applications as either malware (i.e. abnormal) or goodware (i.e. normal) in real time. With this feature, the proposed classifier can effectively detect Android malware applications where other types of classifiers are unsuitable. The EHNFC is based on a modified version of the fast one-pass evolving clustering method (ECM) proposed by [8]. This modification involves changing the process by which the cluster centres and radii are updated, in order to obtain an optimum size and number of clusters and also enhance the classification accuracy of the proposed EHNFC by using high-quality fuzzy rules.

This research makes the following contributions:

- The proposed EHNFC dynamically creates and evolves high-quality fuzzy rules to perfectly predict the newly generated malware based on mining the previous fuzzy rules.
- An EHNFC for obfuscated Android malware detection is proposed.
- A modified version of the ECM is utilized by the classifier.
- The significance of the evolving neuro-fuzzy inference systems in Android obfuscated malware detection is explored.

This paper is structured as follows: Sect. 2 briefly presents two fuzzy inference systems (FISs) related to the proposed EHNFC, namely adaptive fuzzy inference system (ANFIS) and dynamic evolving fuzzy inference system (DENFIS). Section 3 describes the proposed EHNFC including the dataset, feature extraction process, feature selection process and the modified evolving clustering algorithm for enhancing the malware detection in Android-based smartphones. The experimental results and discussion including the comparison method, dataset preparation, metrics of performance and the analysis of the results are presented in Sect. 4. Related work is described in Sect. 5. Finally, conclusions and future work are presented in Sect. 6.

2 Fuzzy inference systems

Neuro-fuzzy systems merge the advantages from fuzzy logic and artificial neural networks, expanding the potential of neural networks [9–13]. Recent advances in the field of neuro-fuzzy techniques include the evolving neuro-fuzzy systems, which integrate the adaptive and evolving learning capability of a neural network with the estimated reasoning and significant interpretation of fuzzy rules. The main characteristic of evolving neuro-fuzzy systems is the ability of the rule base to evolve with adaptive parameters [14]. Several recent publications report a growing number of applications for evolving intelligent systems as a result of their ability to solve many real-world problems in terms of classification, prediction and control [15, 16]. Many FISs are successfully utilized in numerous fields, including the information security field and other applications such as monitoring systems. The success and accuracy of

classification depends mainly on the classifier and the feature selection method used to select potentially useful features, and this holds especially in the field of malware classification. This section presents two FISs related to the proposed EHNFC, i.e. ANFIS and DENFIS.

2.1 Adaptive neuro-fuzzy inference system (ANFIS)

ANFIS integrates a neural network with fuzzy logic, in order to benefit from the advantages of using both, to improve the predictive capabilities [17]. The FIS used by ANFIS consists of three parts: a fuzzy rule base, a reasoning mechanism and a database. The fuzzy rule base includes a set of if–then fuzzy rules. The reasoning method implements the inference process, and the database explains the membership functions in the fuzzy rules [10]. On the other hand, ANFIS utilizes the neural network to fine-tune the membership function and related parameters that approach the preferred datasets [18].

Considering FIS two inputs, x and y, and one output, z. Then, the fuzzy if–then rule based on Takagi and Sugeno [19] is defined as follows:

$$\text{If } x \text{ is } A_i \,\&\, y \text{ is } B_i, \text{ then } \quad f_i = c_i x + \mathrm{d}_i y + r_i \tag{1}$$

where linear c, d and r are the so-called consequent parameters [20]. ANFIS architecture contains five layers, as shown in Fig. 1.

Layer 1. The nodes in this layer utilize the membership functions to produce the membership values based on the inputs.
Layer 2. The firing strength of the rule is the output of each node in this layer.
Layer 3. In this layer, the nodes are fuzzy rules of Takagi–Sugeno type. Each node obtains inputs from the corresponding node in layer 1 and computes their multiplication to find the firing strength of the fuzzy rule r_i:

$$w_i = t(\mu_i((x_{1i}), \mu_i(x_{2i})) = \mu_i(x_{1i}) \cdot \mu_i(x_{2i}) \tag{2}$$

where $\mu_i((x_{1i})$ and $\mu_i(x_{2i}))$ denote the fuzzy membership values of the inputs x_{1i} and x_{2ii}, respectively.
Layer 4. In this layer, the nodes compute the normalization of the rule's firing strength from the corresponding node in the layer 3.
Layer 5. The single node performs the computation for the all outputs from the layer 4.

2.2 Dynamic evolving neuro-fuzzy inference system (DENFIS)

DENFIS [8] is a FIS based on an online clustering to facilitate online learning. DENFIS uses the ECM. ECM is a quick and one-pass clustering method based on the maximum distance. The ECM partitions the input data space into clusters, which are determined by the centres and radii. The number of clusters created by the ECM is estimated dynamically based on a threshold *Dthr*. This threshold value is initially tuned, and it controls the longest length between a data sample of a cluster and the cluster's centre. When the clustering process is complete, a fuzzy rule is generated for each of the clusters. Then, a back-propagation method is used to optimize the generated fuzzy rules. For each prediction, the significant rules are selected to obtain the output.

3 Proposed evolving hybrid neuro-fuzzy classifier

The EHNFC uses a modified ECM to generate evolving fuzzy rules in online mode in order to classify the Android app file (.apk) as either malware (i.e. abnormal) or goodware (i.e. normal). Figure 2 shows the steps for the proposed approach.

As shown in Fig. 2, the proposed EHNFC selects the most significant permissions extracted from the Android APK files and uses them as input to the modified version of the evolving clustering method, to create and evolve the

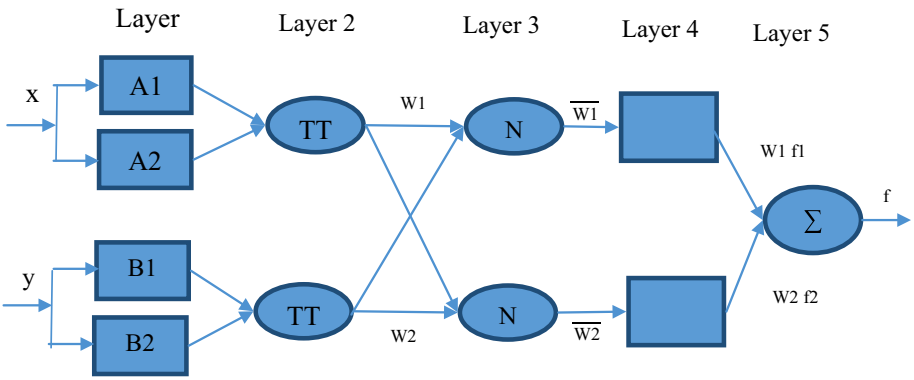

Fig. 1 ANFIS architecture

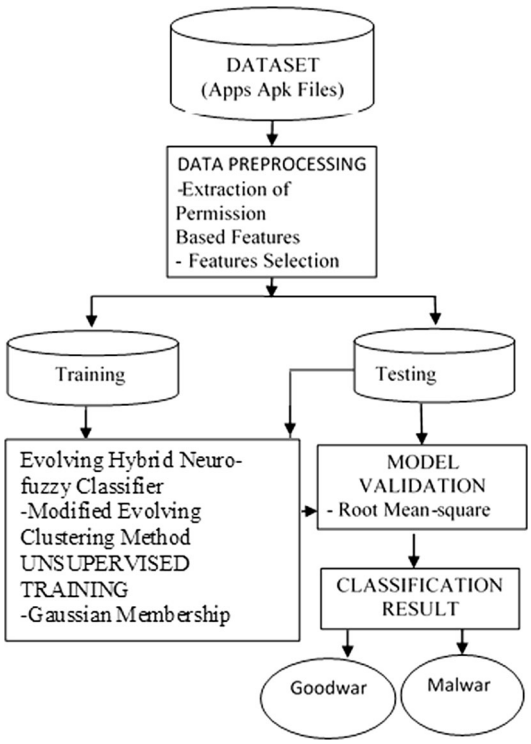

Fig. 2 Proposed evolving hybrid neuro-fuzzy classifier for Android malware detection

fuzzy rules for the discrimination between the goodware and malware apps.

3.1 Dataset and data pre-processing

3.1.1 Dataset

To train and test my proposed EHNFC, I used a dataset provided by the GNOME project [21]. The researchers of this project collected 1200 malware samples, representing different types of existing Android malware families.

The GNOME project made this dataset available to researchers to assist in developing efficient techniques for Android malware detection [22]. I selected 250 malware apps from the GNOME dataset and downloaded 250 goodware apps from official Android application stores as given in Table 1. Each sample (whether malware or goodware) has 50 features and each feature is a permission request; for example, RECEIVE_SMS, INSTALL_PACKAGES, CAMERA and INTERNET are features. If an application uses a feature, the feature is assigned the value 1; otherwise, it is assigned the value 0.

Table 1 Used dataset

	Malware apps	Goodware apps	Total
Training	200	200	400
Testing	50	50	100
Total number of apps	500		

3.1.2 Permission-based feature extraction

The Android operating system supports the development of applications with a wide range of APIs, permitting applications to use the mobile's hardware, like storage, network access and mobile device settings. Moreover, permissions work by controlling access to essential APIs that cost the user money (e.g. voice calls and SMS) and other APIs that risk violating the user's privacy or disabling smartphone services [23]. Access to Android's security APIs is governed by the application-permission system at the time that the application is installed. Each application must declare in advance the permissions it needs, and the user is informed during the installation about the used permissions. If a user prefers to deny the application certain permissions, the installation can be cancelled [24–26]. The requested permissions by Android apps are significant indicators to decide whether the app is malware or goodware [27]. The Android permissions are classified into two main categories, dangerous and normal:

Dangerous permissions enable the Android app to access the user's private information. Normal permissions do not threat the privacy of the user. The examples for the dangers permissions include: READ_CONTACTS, WRITE_-CONTACTS, READ_PHONE_STATE, CALL_PHONE, SEND_SMS and READ_EXTERNAL_STORAGE. The examples for the normal permissions include: INTERNET, ACCESS_NETWORK_STATE, CHANGE_NETWORK_-STATE, MODIFY_AUDIO_SETTINGS, REORDER_-TASKS and SET_TIME_ZONE.

To extract the permissions from Android applications, I used Apktool to decompress the Android application package file (.apk). The decompression of the '.apk' results in three types of content, namely 'AndroidManifest.xml', 'Classes.dex' and the 'res' folder. 'AndroidManifest.xml' is an XML file that contains the application permissions.

3.1.3 Feature selection

In neuro-fuzzy systems, such as the proposed EHNFC, the big amount of the features, many of which are unrelated or redundant, causes many problems, like confusing the learning algorithm, over-fitting and reducing the classification accuracy. These undesirable effects are even more problematic when applying neuro-fuzzy systems to mobile devices, because they are constrained by low processing, limited storage and low battery power. Therefore, it is significant to use a suitable feature selection method at the preparatory stage to enable the proposed EHNFC to detect

the malware efficiently—with a faster classification process and a lower classification time.

I used the information gain ratio (IGR) method [28] to select the most significant features in Android malware. The IGR method works by extracting the similarities between sets of Android application features and then calculating and scoring each feature individually. The IGR is derived as follows:

$$\text{gain\_}r(X,Y) = \frac{\text{gain}(X,Y)}{\text{split\_info}(Y)} \tag{3}$$

$$\text{split\_info}(Y) = \sum_i \left(\frac{|Y_i|}{|Y|}\right) \log \frac{|Y_i|}{Y} \tag{4}$$

where gain_r (X, Y) denotes the gain ratio which is the number of occurrences of feature X in class Y. Y_i and $|Y_i|$ are the number of occurrences of feature X in class Y_i, the ith subclass of Y, and the number of features in Y_i, respectively. The selected features are shown in Fig. 3, along with the scores resulting from the IGR algorithm.

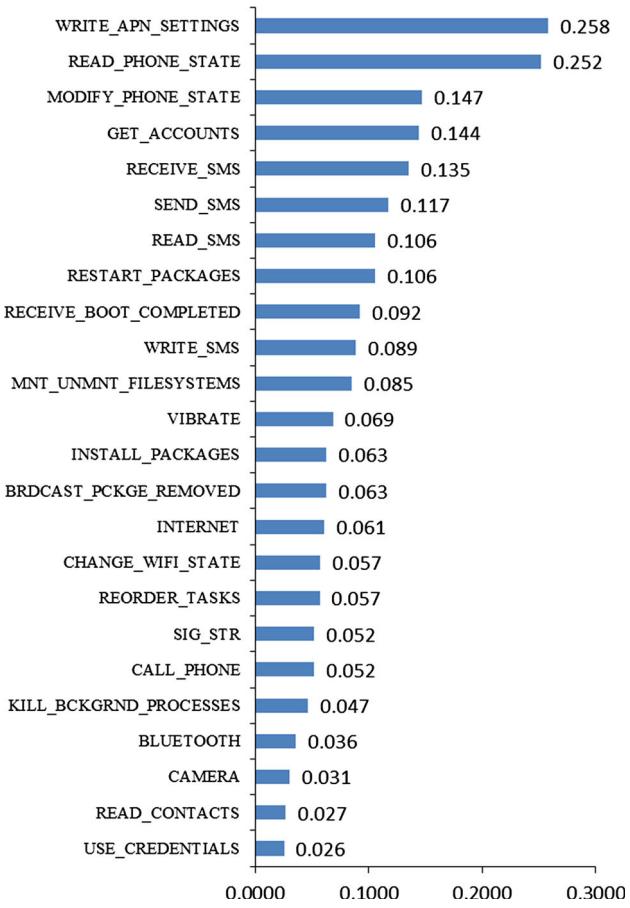

Fig. 3 Names and scores for selected permission-based features

3.2 Modifying the evolving clustering method

To enable the proposed EHNFC to create and evolve the fuzzy rule base in online mode, a reliable data clustering algorithm is needed. The ECM is a quick algorithm for online clustering, based on the maximum distance. ECM creates and evolves the fuzzy rules by partitioning the input data in online mode. For each cluster, the maximum distance *MaxDst* between the sample input and the centre of the cluster is smaller than a certain threshold *Dthr*—a parameter that influences the number of generated clusters. At the beginning of the clustering process, the set of clusters is empty. With the creation of a new cluster, the position of the current input sample is defined as the cluster centre *Cc*, and its radius is firstly set to zero. When new input samples are provided, the previous clusters are updated by incrementing their radii and changing theirs cluster centres [15].

With the ECM, the update of cluster centres occurs only when a new data sample that does not fit in any cluster is included in a particular cluster. If the new data sample fits in an existing cluster, the cluster centre will not be updated. As a result, cluster convergence is decreased. This paper proposes modifying the ECM to enhance cluster convergence and to better represent the clusters. The modification is as follows: for each cluster, the maximum distance *MaxDst* between the sample data input and the cluster centre is set to be less than the sum of a threshold *Dthr* and the standard deviation *Std* of data samples in the cluster. The modified ECM uses the standard deviation of data samples in the cluster to reveal dispersions within clusters as a measure of cluster convergence. Thus, the standard deviation provides information regarding the relative dispersion of data samples in the clusters. By incorporating standard deviation in the modified ECM, clusters of different sizes with concentrated centres can be created. The following details the steps to the modified ECM:

Step 1: Read the input sample x_i and create the first cluster C_i with centre Cc_i and radius $Ra_i = 0$, where $i = 1, 2, 3, \ldots, n$ data samples.

Step 2: If all inputs are handled, the algorithm ends. Otherwise, read the next input sample $\boldsymbol{x}_i$ and calculate the normalized Euclidian distance D_{ij} between this sample and all n cluster centres Cc_j that exist already. Thus, $D_{ij} = \|\boldsymbol{x}_i - Cc_j\|, j = 1, 2, \ldots, n$ is computed. Then, calculate the standard deviation *Std* of cluster C_j using the following equation:

$$\text{Std}\, c_j = \sqrt{\frac{\sum_{i \in C_j} d\left(x_i, Cc_j\right)^2}{Nc_j}} \tag{5}$$

where d(x_i ,Cc_j) denotes the length between the centre of cluster C_j and data point x_i and Nc_j is the total number of data samples in cluster C_j.

Step 3: In case there is a centre of cluster Cc_j and the value of distance $D_{ij} = \| \boldsymbol{x}_i - Cc_j \|$ is equivalent to or smaller than the radius Ru_j, then the current data sample fits the cluster C_m with the least of these distances: $D_{ik} = \|\boldsymbol{x}_i - Cc_k\| = \min \ (\| \ \boldsymbol{x}_i - Cc_j \ \|)$, where $D_{ij} <= Ru_j$, $j = 1, 2, \ldots, n$

In this situation, there is no creation of new clusters or updates of existing clusters, and the algorithm proceeds to Step 2; otherwise, the algorithm proceeds to the next step.

Step 4: Locate a cluster C_b (based on its centre Cc_b and radius Ru_b) using all n already existing cluster centres by computing $S_{ij} = D_{ij} + Ru_j$, $j = 1, 2, \ldots, n$ and then selecting the cluster centre Cc_b with the least value S_{ib}: $S_{ib} = D_{ib} + R_{ub} = \min \{S_{ij}\}, j = 1, 2, \ldots, n$.

Step 5: If $S_{ib} > 2 \times Dthr + Stdc_j$, the data point x_i does not appropriate to all available clusters. A new cluster is generated as explained in Step 1, and the algorithm proceeds to Step 2. Otherwise, the cluster C_b is restructured by shifting its centre, Cc_b, and growing the value of its radius, Ru_b. The updated radius Ru_b is assigned the value $S_{ib}/2$, and the new centre Cc_b is placed on the path linking the cluster centre Cc_b and the new input vector x_i. Thus, the space between the new centre Cc_b and the sample x_i is equal to the updated Ru_b. The algorithm then proceeds to Step 2.

3.3 Generating fuzzy rules

The EHNFC utilizes a fuzzy inference engine consisting of a number of fuzzy as in [19]. Fuzzy rules are presented in the form: 'IF antecedent, THEN consequent'. TS-type rules are first-order rules that include linear functions as the consequent. The general form for the rule is explained as follows:

$$\text{IF } M \text{ is } S_1, \text{ AND } N \text{ is } S_2 \text{ THEN } Z = \beta_0 + \beta_1 X + \beta_2 Y \qquad (6)$$

where M and N are the variables of the antecedent, S_1 and S_2 are the fuzzy sets and β_0, β_1 and β_2 are linear parameters.

In order to generate the fuzzy rules, the EHNFC creates the membership functions from the clusters produced by the modified ECM. The membership function defines the data samples in the input dataset as a membership degree in the range [0,1]. There are many categories of membership functions, and the selection of a membership function is generally decided by experts or chosen depending on its appropriateness in terms of convenience, simplicity, efficiency and speed [17]. In the proposed EHNFC, I used the Gaussian membership function because of its convenience and simplicity. The Gaussian membership function is explained as follows:

$$\mu_i = \exp\left[-\frac{(z_i - v)^2}{2\delta^2}\right] \qquad (7)$$

where $z_i (i = 1,2,\ldots,N)$ are the data points in cluster z and δ and v are the standard deviation and mean of the data points in cluster z, respectively.

The EHNFC uses the modified ECM to partition the inputs into clusters in order to create the TS-type fuzzy rules. The modified ECM generates fewer big clusters, and this enables the EHNFC to generate rules that are more tailored to the input data, improving the classification accuracy of the EHNFC.

4 Experimental results and discussion

To evaluate the performance of the proposed EHNFC, two comparisons were made. First, the well-known FISs ANFIS and DENFIS were used for comparison [8, 29], because they use the same fuzzy rule type as the proposed EHNFC. Second, the proposed approach is compared with state-of-the-art malware detection approaches including our previous research k-ANFIS [30], RiskRanker [31], DroidMOSS [1], Dendroid [23] and Drebin [32].

4.1 Comparison with the fuzzy inference systems

DENFIS uses the standard ECM for clustering, and thus, it is useful for comparison with the proposed EHNFC. The definition of the membership function (MF) and its corresponding value is an important factor in the development of a neuro-fuzzy inference system [23]. Four different MFs were used to build four ANFIS models, viz. the triangular MF (TRIMF), trapezoidal MF (TRAP), generalized bell MF (GBELL) and pi-shaped MF (PIMF). The performance of the EHNFC was compared with the four ANFIS models and DENFIS by using the Android malware dataset from the GNOME project [21]. First, I split the dataset into a testing dataset and a training dataset. To prepare the testing and training datasets, K-fold cross-validation technique was used [33]. The dataset was randomly divided into five sub-datasets of the same size, and the EHNFC was trained five times. Each time, I used fourfold for training and onefold for testing. Because the accuracy of the overall cross-validation influenced by the random allocation of classes into five distinct folds, I employed a stratification process, aiming to create folds containing the same percentage of classes (malware and goodware) to ensure results with lower variance and lower bias [34]. To

measure the accuracy of the proposed EHNFC, I used the root mean-square error (RMSE) and the non-dimensional error index (NDEI). The RMSE measures the difference between values implied by the EHNFC and the observed values:

$$\mathrm{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(M_i - P_i)^2} \tag{8}$$

where n denotes the number of samples, M_i is the observed value and P_i is the predicted value. The NDEI is calculated by dividing the RMSE by the standard deviation.

$$\mathrm{NDEI} = \frac{\mathrm{RMSE}}{\mathrm{Stdev}(M)} \tag{9}$$

where Stdev(M) denotes the standard deviation of the observed values.The accuracy of the cross-validation is computed by finding the average of the n individual accuracy measures:

$$X = \frac{1}{n}\sum_{i=1}^{n} B_i \tag{10}$$

where X is the accuracy of cross-validation, n denotes the used folds number and B_i is each fold accuracy measure.

Figure 4 shows the effectiveness of the proposed EHNFC, using a modified version of the ECM. The results illustrate that the EHNFC is considerably more accurate than the ANFIS models (TRIMF–ANFIS, TRAP–ANFIS, GBELL–ANFIS, PIMF–ANFIS) and DENFIS in terms of malware classification, insofar as it achieved a lower RMSE and NDEI than that of the ANFIS models and DENFIS in all cases.

To assess the performance of the proposed EHNFC, a confusion matrix [35] was used to derive the following measures for Android malware classification evaluation.

True positive (TP): malware is correctly classified.
False positive (FP): malware is incorrectly classified as goodware.
False negative (FN): goodware is incorrectly classified as malware.
True negative (TN): goodware is correctly classified.

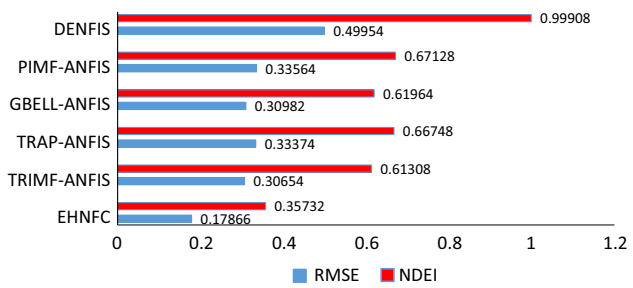

Fig. 4 Comparison of the results obtained with the proposed EHNFC, ANFIS models and DENFIS

A more detailed analysis of the performance of the proposed EHNFC is based on the following:

- Accuracy (ACC): the proportion of the total number of malware variants that are correctly classified. This is needed to determine the ultimate success of proposal.

$$\mathrm{Accuracy} = \frac{(\mathrm{TP} + \mathrm{TN})}{(\mathrm{TP} + \mathrm{FN} + \mathrm{TN} + \mathrm{FP})} \tag{11}$$

  Error rate (ER): the proportion of the total number of wrong classifications. Here, ER = 1 – ACC.
- Receiver operating characteristics (ROC): used to assess the performance of classifiers based on the rate of TPs and FPs. It is represented as a graph in which the TP rate is displayed along the Y-axis and the FP rate is displayed along the X-axis. To compare the performance of classifiers, a significant measure of the accuracy is used, namely the area under the ROC curve (AUC) [36]. The value of the AUC is between 0.5 and 1. If the AUC is equal to 1.0, the performance of the classifier is 100% accurate, as both TP rate and FP rate are equal to 1.0.

Table 2 shows the false positive rate, true positive rate, accuracy and the AUC for the proposed EHNFC, ANFIS models and DENFIS. The table shows that the proposed EHNFC achieved the highest accuracy and the lowest false positive rate among all the tested classifiers, conferring the EHNFC with a considerable advantage. The EHNFC's AUC is slightly higher than the ANFIS models and DENFIS, which also indicates that the proposed EHNFC has the highest accuracy. The AUC achieved with all classifiers was higher than 0.9, with the exception of the PIMF–ANFIS classifier, which achieved the lowest AUC.

4.2 Comparison with the recent comparable malware detection approaches

Although EHNFC demonstrates a better performance compared to other neuro-fuzzy inference systems, in the end it has to be compared with most closely related approaches. Consequently, I compare it against nine similar malware

Table 2 Accuracy, false positive rate, true positive rate and area under the curve of the proposed EHNFC, ANFIS models and DENFIS

Classifier	Accuracy	TP	FP	AUC
EHNFC	0.900	0.8824	0.05	0.950
TRIMF–ANFIS	0.880	0.7895	0.1111	0.930
TRAP–ANFIS	0.865	0.8235	0.1429	0.918
GBELL–ANFIS	0.838	0.8235	0.15	0.913
PIMF–ANFIS	0.784	0.8421	0.3333	0.788
DENFIS	0.822	0.875	0.1905	0.920

Table 3 Comparison between the proposed EHNFC and other comparable malware classifiers

Classifier	Used features	Used dataset	detection method	Classification accuracy
K-ANFIS [30]	Permission used in Android application	Google Play and Android malware genome project	Fuzzy inference with *k*-means	FP = 10%
Apposcopy [44]	Android applications characteristics such as control flow and data flow	Android malware genome project	Hybrid of inter-component call graph and static taint analysis	FN = 10%
DroidMOSS [1]	Instructions used in the app and the information about the author	Several Android markets	Fuzzy hashing technique	FP = 7.1–13.3%
DREBIN [32]	Permissions, hardware access, API calls and network addresses	Several markets and Android malware genome project	SVM	FP = 6%
MADAM [38]	CPU and memory usage statistics. System calls in Android app.	Several Android markets	K-nearest neighbour (K-NN) algorithm	FP = 5%
Crowdroid [37]	System calls	VirusTotal	*k*-means clustering	FP = 20%
RiskRanker [31]	various information from Android system and application	Several Android markets	Multiple malware behaviour signatures	FN = 9%
Dendroid [23]	Control-flow graph and code structure in Android app	Android malware genome project	Text mining techniques and information retrieval methods	FP = 5.74%
AndroSimilar [46]	Statistic-based characteristics of the Android app	Google Play, Android malware genome project	Statistical signature-based method	FP = 7%
Andromaly [48]	Size of used memory and CPU load	Dataset collected from Google Play	*k*-means clustering, Bayes nets, decision tree and naive Bayes	FP = 10%
My proposed approach EHNFC	Permission used in Android application	Google Play and Android malware genome Project	Fuzzy inference with evolving clustering method	FP = 5% FN = 5%

classifiers. The results of the comparisons are presented in Table 3. The classification accuracy of the classifiers varies considerably. The proposed classifier EHNFC achieved the highest performance and detects the malware with minimum false positive of 5%, while the classifier Crowdroid [37] achieved the lowest performance and detects the malware with false positive of 20%, and also the results show that the false positive rate of EHNFC (5%) is slightly better than the false positive rate of Dendroid (5.74%) and that gives EHNFC an important advantage. MADAM [38] achieved the same of the proposed classifier EHNFC. However, MADAM has been tested using small dataset consisting of 10 malware apps and 50 goodware apps.

Table 3 also shows that the proposed EHNFC achieved the lowest false negative of 5% when compared with the RiskRanker classifier that achieved highest false negative of 9%. However, the ability of the proposed classifier EHNFC to evolve its structure by learning new malware detection fuzzy rules improves its detection accuracy and proves that it is more effective than other classifiers.

5 Related work

Android malware represents real threats for both users and developers. The malware threats cover wide range including stealing personal and important information and sending short message services (SMS) which cause undesired billing, abuse of smartphone resources and denial of services [22, 39–41]. To detect malware and advance the security and privacy of smartphones, a number of approaches have been proposed. The malware detection approaches are categorized as follows:

5.1 Signature-based malware detection

Many malware detectors like antivirus are signature based. Signature-based detectors can classify Android application as malware if the sequence of instructions in the application is matched by a predefined fixed expression. The simplest form of the signatures is a series of instructions and bytes [42]. Signature-based malware detection has been studied in many researches including DroidMOSS [1], Riskranker [31], Kirin [43], Apposcopy [44] and AppGuard [45].

In [1], DroidMOSS applies fuzzy hashing to distinguish legitimate repackaged apps in third-party Android marketplaces, by computing the degree of similarity between plaintiff and repackaged applications. The results from DroidMOSS's experiments indicate that between 5 and 13% of apps hosted on third-party Android markets are repackaged. However, obfuscated malware is able to modify the sequence of executing the methods and classes to evade the DroidMOSS and reduce its accuracy. DroidMOSS reports false positive rate from 7.1% up to 13.3%.

In the work performed by Faruki, Laxmi, Bharmal, Guar and Ganmoor [46], AndroSimilar was proposed for detecting

Android malware apps using a statistical signature-based method. Testing results showed that AndroSimilar detected variants of well-known malware types with accuracy of 72.3%, and the authors expressed plans for refining AndroSimilar for deployment as a smartphone antivirus program. However, AndroSimilar depends on the syntactic similarity of the application's file rather than using the semantic similarity, which reduces the classification accuracy and generates false positive rate up to 7%.

The AppGuard system proposed by Backes et al. [45] imposes user-defined security strategies on applications without modifying the smartphone's firmware and without the need for root access. Their test results showed that AppGuard uses little space and generates low runtime overhead. A limitation of this approach is that AppGuard monitors only Java calls in the application and ignores the function calls in the native libraries. Thus, it is not able to detect obfuscated malware.

Riskranker [31] classifies the Android applications based on the extracted behaviour signatures as high, medium or low risk. The extracted behaviour signatures include the used class path and sending SMS from background. However, obfuscated malware can easily modify the opcode class path and the false negative rate generated by RiskRanker is 9%.

Apposcopy is a semantics-based approach for malware classification; it integrates the static taint analysis with the Inter-Component Call Graph of the application. Apposcopy detects the Android malware with rate 90% and false negative rate of 10%.

According to many recent studies, signature-based malware detectors have many limitations; it can be easily evaded and defeated using simple application obfuscations [37, 47]. Therefore, the malware signatures used by the detector need to be updated continuously as new version of the same malware appears; if the malware authors distribute a new previously unseen malware, the malware detector becomes useless and will not be able to detect the malware [48].

In comparison, my proposed EHNFC utilizes its high-quality fuzzy rules to classify the Android malware with high accuracy and low false negative of 5% and false positive of 5%. The results also demonstrate that my proposed EHNFC achieved better classification in comparison with state-of-the-art Android malware classification approaches, and it has the capability of evolving its structure by learning new malware detection fuzzy rules to improve its detection accuracy.

5.2 Behaviour-based malware detection

In the behaviour-based malware detection, the various features of the application like the Android permissions and API calls are extracted and used as input to machine learning algorithms which perform the classification [49]. Behaviour-based malware detection has been considered in many studies including our previous research k-ANFIS [30], MADAM [38], DREBIN [32], DroidAPIMiner [37, 50], Dendroid [23] and TaintDroid [51].

In our previous research k-ANFIS [30], we explore the feasibility of using neuro-fuzzy techniques for Android detection based on application's permissions and we present an adaptive neuro-fuzzy inference system to classify the Android apps into malware and goodware. The results illustrate that the proposed classifier is effective in Android malware detection. The results also emphasize that the neuro-fuzzy techniques are feasible to employ in the field. The limitation of k-ANFIS is that it is not able to evolve and generate new fuzzy rules to accurately classify the apps and also it can be evaded by code obfuscation techniques. k-ANFIS reported false positive rate of 10%.

TaintDroid [51] is a behaviour-based approach which monitors the applications in a controlled environment to secure private data and detect malware apps by tracking the information flow, using the dynamic taint analysis mechanism. TaintDroid is not able to detect control flows. Thus, it can be evaded by code obfuscation based on control dependencies.

A multi-level anomaly detector for Android malware (MADAM) is proposed by Dini et al. [38]. MADAM periodically monitors certain features at the kernel and user levels to detect app behaviour and classify apps as goodware or malware. The collected samples are classified based on a K-nearest neighbour (K-NN) algorithm. The detection rate of MADAM is 93 with 5% false positive.

DREBIN [32] utilizes machine learning methods to classify Android malware application based on set of features. The used features include the permissions used by application, API calls and network address. The detection rate of DREBIN is 94 with 6% false positive. DREBIN could be evaded by malware that implements the obfuscation technique and the dynamic load of code.

Aafer et al. [50] proposed DroidAPIMiner, a lightweight classifier for Android malware classification based on the monitored malware behaviour at the API level. Their evaluation of this proposal showed that the classifier reports a false positive rate of 2.2%. DroidAPIMiner is not accurate when used to detect malware applications that depend on permissions not available in the training dataset.

The limitations of the behaviour-based detectors include the high percentages of false positives, high runtime overhead and consuming the limited resources (memory, CPU and power source) of mobile devices [48], and it can be easily evaded using code obfuscations techniques. My proposal overcomes these problems by using a fast one-pass algorithm to evolve its structure and

learn new fuzzy rules to detect malware and improve its detection accuracy.

6 Conclusion

This paper proposed EHNFC for Android malware classification. The proposed EHNFC is based on a modified version of the evolving clustering method. This modified algorithm partitions selected permission-based features into clusters, generating larger and fewer clusters than the unmodified algorithm. This enables the EHNFC to generate rules that are better suited to the input data, improving the accuracy of the classification into malware and goodware and dynamically evolving its knowledge base. The experimental results show that the proposed EHNFC achieved higher accuracy than the other neuro-fuzzy approaches, namely ANFIS and DENFIS. In comparison with recent malware detection approaches, the proposed classifier EHNFC achieved the highest performance and detects the malware with minimum false positive of 5% and false negative of 5%.

The combination of the proposed classifier with the dynamic analysis of Android apps will be considered in future work. The goal of the combination with the dynamic analysis is to gain more knowledge and get insights on most significant features and to differentiate between malware and goodware, with high accuracy and minimum run-time overhead, and these are expected to improve the performance of the proposed classifier. Moreover, I shall test and evaluate the proposed classifier on different platforms, including desktop systems.

Acknowledgements This work was supported by the Deanship of Scientific Research (DSR), King Abdulaziz University, Jeddah, Saudi Arabia, under Grant No. (830-863-D1435). The author, therefore, gratefully acknowledges the DSR technical and financial support.

Compliance with ethical standards

Conflict of interest The author declares that he has no conflict of interest.

References

1. Zhou W, Zhou Y, Jiang X, Ning P (2012) DroidMOSS: detecting repackaged smartphone applications in third-party Android marketplaces. Paper presented at the Proceedings of the second ACM conference on data and application security and privacy. ACM, New York, pp 317–326. doi:10.1145/2133601.2133640
2. Gartner. Inc. (2014) Android to surpass one billion users across all devices in 2014. Retrieved June, 17, 2016, from http://www.gartner.com/newsroom/id/2645115
3. Google (2015) Official android market; Google Play. Retrieved Feb 17, 2016, from https://market.android.com
4. Oberheide J, Miller C (2012) Dissecting the Android bouncer. Presented at SummerCon2012, New York
5. Kaspersky Lab (2015) IT threat evolution report for Q1 of 2015. Retrieved June 17, 2016, from http://usa.kaspersky.com/about-us/press-center/in-the-news/mobile-threats-rise-q1-2015-report-shows-sc-magazine
6. Felt AP, Finifter M, Chin E, Hanna S, Wagner D (2011) A survey of mobile malware in the wild. In: Jiang X, Bhattacharya A, Dasgupta P, Enck W (eds) On the 1st ACM workshop on security and privacy in smartphones and mobile devices. ACM, New York, pp 3–14. doi:10.1145/2046614.2046618
7. Elhadi AAE, Maarof MA, Barry BI, Hamza H (2014) Enhancing the detection of metamorphic malware using call graphs. Comput Secur 46:62–78. doi:10.1016/j.cose.2014.07.004
8. Kasabov N, Song Q (2002) DENFIS: dynamic evolving neural-fuzzy inference system and its application for time-series prediction. IEEE Trans Fuzzy Syst 10(2):144–154. doi:10.1109/91.995117
9. Rutkowski L (2008) Computational intelligence: methods and techniques. Springer, Berlin
10. Jang J-S, Sun C-T, Mizutani E (1997) Neuro-fuzzy and soft computing: a computational approach to learning and machine intelligence. Prentice Hall, Upper Saddle River
11. Mumford CL, Jain LC (2009) Computational intelligence. Springer, Berlin
12. Kruse R, Borgelt C, Klawonn F, Moewes C, Steinbrecher M, Held P (2013) Computational intelligence: a methodological introduction. Springer, Berlin
13. Wang L-X, Mendel JM (1993) Fuzzy basis functions, universal approximation, and orthogonal least squares learning. IEEE Trans Neural Netw 3(5):807–814. doi:10.1109/72.159070
14. Kasabov N (2013) The evolution of the evolving neuro-fuzzy systems: from expert systems to spiking-, neurogenetic-, and quantum inspired. In: Seising R, Trillas E, Moraga C, Termini S (eds) On fuzziness. Springer, Berlin, pp 271–280. doi:10.1007/978-3-642-35641-4_41
15. Angelov P, Filev DP, Kasabov E (eds) (2010) Evolving intelligent systems: methodology and applications. Wiley, Hoboken
16. Almomani A (2016) Fast-flux hunter: a system for filtering online fast-flux botnet. Neural Comput Appl. doi:10.1007/s00521-016-2531-1
17. Jang J-SR (1993) ANFIS: adaptive-network-based fuzzy inference system. IEEE Trans Syst Man Cybern 23(3):665–685. doi:10.1109/21.256541
18. Wu J-D, Hsu C-C, Chen H-C (2009) An expert system of price forecasting for used cars using adaptive neuro-fuzzy inference. Expert Syst Appl 36(4):7809–7817. doi:10.1016/j.eswa.2008.11.019
19. Takagi T, Sugeno M (1985) Fuzzy identification of systems and its applications to modeling and control. IEEE Trans Syst Man Cybern SMC-15(1):116–132. doi:10.1109/tsmc.1985.6313399
20. Ubeyli ED (2008) Adaptive neuro-fuzzy inference system employing wavelet coefficients for detection of ophthalmic arterial disorders. Expert Syst Appl 34(3):2201–2209. doi:10.1016/j.eswa.2007.02.020
21. Zhou Y, Jiang X (2012) Android malware genome project. Retrieved Dec 15, 2014, from http://www.malgenomeproject.org/
22. Zhou Y, Jiang, X (2012) Dissecting Android malware: characterization and evolution. Paper presented at the 2012 IEEE symposium on security and privacy. IEEE, San Francisco, pp 95–109. doi:10.1109/sp.2012.16
23. Suarez-Tangila G, Tapiadora JE, Peris-Lopeza P, Blasco J (2014) Dendroid: a text mining approach to analyzing and classifying code structures in Android malware families. Expert Syst Appl 41(4):1104–1117. doi:10.1016/j.eswa.2013.07.106

24. Felt AP, Chin E, Hanna S, Song D, Wagner D (2011) Android permissions demystified. In: Chen Y, Danezis G, Shmatikov V (eds) On the 18th ACM conference on computer and communications security. ACM, New York, pp 627–638. doi:10.1145/2046707.2046779
25. Fang Z, Han W, Li Y (2014) Permission based Android security: issues and countermeasures. Comput Secur 43:205–218. doi:10.1016/j.cose.2014.02.007
26. Xiong P, Wang X, Niu W, Zhu T, Li G (2014) Android malware detection with contrasting permission patterns. China Commun 11(8):1–14. doi:10.1109/cc.2014.6911083
27. Jang JW, Kang H, Woo J, Mohaisen A, Kim HK (2016) Andro-dumpsys: anti-malware system based on the similarity of malware creator and malware centric information. Comput Secur 58:125–138. doi:10.1016/j.cose.2015.12.005
28. Mori T (2002) Information gain ratio as term weight: the case of summarization of IR results. Paper presented at the proceedings of the 19th international conference on computational linguistics (COLING '02). Association for Computational Linguistics, Stroudsburg, pp 1–7. doi:10.3115/1072228.1072246
29. Singh R, Kainthola A, Singh TN (2012) Estimation of elastic constant of rocks using an ANFIS approach. Appl Soft Comput 12(1):40–45. doi:10.1016/j.asoc.2011.09.010
30. Abdulla S, Altaher A (2015) Intelligent approach for android malware detection. KSII Trans Internet Inf Syst 9(8):2964–2983. doi:10.3837/tiis.2015.08.012
31. Grace M, Zhou Y, Zhang Q, Zou, S, Jiang X (2012) Riskranker: scalable and accurate zero-day android malware detection. In: Davies N, Seshan S, Zhong L (eds) On the 10th international conference on Mobile systems, applications, and services. ACM, New York, pp 281–294. doi:10.1145/2307636.2307663
32. Arp D, Spreitzenbarth M, Hübner M, Gascon H, Rieck K, Siemens CERT (2014) Drebin: effective and explainable detection of android malware in your pocket. In: Bauer L (ed) On the annual symposium on network and distributed system security (NDSS). doi:10.14722/ndss.2014.23247
33. Fushiki T (2011) Estimation of prediction error by using K-fold cross-validation. Stat Comput 21(2):137–146. doi:10.1007/s11222-009-9153-8
34. Katsis CD, Katertsidis N, Ganiatsas G, Fotiadis DI (2008) Toward emotion recognition in car-racing drivers: a biosignal processing approach. IEEE Trans Syst Man Cybern 38(3):502–512. doi:10.1109/tsmca.2008.918624
35. Witten IH, Frank E, Hall MA (2011) Data mining: practical machine learning tools and techniques. Elsevier Science, Burlington
36. Fawcett T (2006) An introduction to ROC analysis. Pattern Recognit Lett 27(8):861–874. doi:10.1016/j.patrec.2005.10.010
37. Burguera I, Zurutuza U, Nadjm-Tehrani S (2011) Crowdroid: behavior-based malware detection system for android. In: Jiang X, Bhattacharya A, Dasgupta P, Enck W (eds) On the 1st ACM workshop on Security and privacy in smartphones and mobile devices. ACM, New York, pp 15–26. doi:10.1145/2046614.2046619
38. Dini G, Martinelli F, Saracino A, Sgandurra D (2012) MADAM: a multi-level anomaly detector for Android malware. In: Kotenko I, Skormin V (eds) Computer network security. Springer, Berlin, pp 240–253. doi:10.1007/978-3-642-33704-8_21
39. Elish KO, Shu X, Yao DD, Ryder BG, Jiang X (2015) Profiling user-trigger dependence for Android malware detection. Comput Secur 49:255–273. doi:10.1016/j.cose.2014.11.001
40. Choo KKR (2011) The cyber threat landscape: challenges and future research directions. Comput Secur 30(8):719–731. doi:10.2139/ssrn.2339821
41. Seo SH, Gupta A, Sallam AM, Bertino E, Yim K (2014) Detecting mobile malware threats to homeland security through static analysis. J Netw Comput Appl 38:43–53. doi:10.1016/j.jnca.2013.05.008
42. Griffin K, Schneider S, Hu X, Chiueh TC (2009) Automatic generation of string signatures for malware detection. In: Kirda E, Jha S, Balzarotti D (eds) Recent advances in intrusion detection. Springer, Berlin, pp 101–120. doi:10.1007/978-3-642-04342-0_6
43. Enck W, Ongtang M, McDaniel P (2009) On lightweight mobile phone application certification. In: Al- Shaer E, Jha S, Keromytis AD (eds) On the 16th ACM conference on computer and communications security. ACM, New York, pp 235–245. doi:10.1145/1653662.1653691
44. Feng Y, Anand S, Dillig I, Aiken A (2014) Apposcopy: semantics-based detection of android malware through static analysis. In: Cheung S, Orso A, Storey M (eds) On the 22nd ACM SIGSOFT international symposium on foundations of software engineering. ACM, New York, pp 576–587. doi:10.1145/2635868.2635869
45. Backes M, Gerling S, Hammer C, Maffei M, von Styp-Rekowsky P (2014) AppGuard: fine-grained policy enforcement for untrusted Android applications. In: Garcia-Alfaro J, Lioudakis G, Cuppens-Boulahia N, Foley S, Fitzgerald MW (eds) Data privacy management and autonomous spontaneous security. Springer, Berlin. doi:10.1007/978-3-642-54568-9_14
46. Faruki P, Laxmi V, Bharmal A, Gaur MS, Ganmoor V (2015) AndroSimilar: robust signature for detecting variants of Android malware. J Inf Secur Appl 22:66–80. doi:10.1016/j.jisa.2014.10.011
47. Rastogi V, Chen Y, Jiang X (2013) Droidchameleon: evaluating android anti-malware against transformation attacks. In: Chen K, Xie Q, Qiu W, Li N, Tzeng W (eds) On the 8th ACM SIGSAC symposium on information, computer and communications security. ACM, New York, pp 329–334. doi:10.1145/2484313.2484355
48. Shabtai A, Kanonov U, Elovici Y, Glezer C, Weiss Y (2012) "Andromaly": a behavioral malware detection framework for android devices. J Intell Inf Syst 38(1):161–190. doi:10.1007/s10844-010-0148-x
49. Zhang M, Duan Y, Yin H, Zhao Z (2014) Semantics-aware Android malware classification using weighted contextual API dependency graphs. In: Ahn G, Yung M, Li N (eds) On the 2014 ACM SIGSAC conference on computer and communications security. ACM, New York, pp 1105–1116. doi:10.1145/2660267.2660359
50. Aafer Y, Du W, Yin H (2013) DroidAPIMiner: mining API-level features for robust malware detection in Android. In: Zia T, Zumaya A, Varadharajan V, Mao M (eds) Security and privacy in communication networks. Springer, Berlin, pp 86–103. doi:10.1007/978-3-319-04283-1_6
51. Enck W, Gilbert P, Han S, Tendulkar V, Chun BG, Cox LP, Jung J, MacDaniel P, Sheth AN (2014) TaintDroid: an information-flow tracking system for realtime privacy monitoring on smartphones. ACM Trans Comput Syst (TOCS) 32(2):5. doi:10.1145/2619091