ISSN XXXX XXXX © 2018 IJESC

Research Article Volume 8 Issue No.3

Iterative Classifier Fusion System Android Malware Detection

A. Syed asmathunnisa[1], M. Nafa fathima[2], D. Kavitha[3]
BE student[1, 2], Assistant professor[3]
Department of Computer Science and Engineering
Dhaanish Ahmed College of Engineering, Chennai, India

Abstract:
Malicious software (malware) poses serious challenges for security if big data. The number and complexity of malware targeting Android devices have been exponentially increasing with the ever growing popularity of Android devices. To address this problem, multi-classifier fusion systems have long been used to increase the accuracy of malware detection for personal computers. However, previously developed systems are quite large and they cannot be transferred to Android platforms. To this end, we propose Iterative Classifier Fusion System (ICFS), which is a system of minimum size, since it applies a smallest possible number of classifiers. The system applies classifiers iteratively in fusion with the new extensive empirical study to determine the best options to be employed in ICFS and to compare the effectiveness of ICFS with several other traditional classifiers. The experiments show that the best outcomes for Android malware detection have been obtained by the ICFS procedure using Lib SVM with polynomials kernel, combined with Multilayer perception and NB tree classifiers and applying IFS feature selection based on Wrapper Subset Evaluator with Particle Swarm Optimization.

Keywords: Android malware, smart phone security, cloud security, big data security, iterative systems, feature selection classifier fusion

I. INTRODUCTION

BIG DATA has become ubiquitous in all aspects of modern society and in various research domains. With mobile devices such as smartphones becoming capable to run complex software equivalent to that of personal computers (PCs), owners are influence their smart phones for a wide variety of applications such as accessing and storing big data that include sensitive and commercial-in-confidence data [1] [2]. Unfortunately, the rapid growth and widespread user acceptance of smartphones have followed with a flow both in number and complexity of malware that target popular mobile phone platforms. With Android possessing the largest share of the mobile phone market, malware specially targeting Android devices have been growing exponentially both in size and sophistication. Malware forms one of the core cybersecurity threat landscape in distributed computing [4]. In big data sphere, malware has been identified as one of the top security and privacy challenges to be tackled [5]. The size and the variety of Android malware seen in the concurrent Android malware detection databases pertains to the domain of big data. Also, as smartphones are commonly used for accessing and storing big data [1], Android malware poses a serious challenge for big data security. Since it can impact the data authentic ability and consequently the system it is important that Android malware are detected early and removed from a system. However, due to the absence of efficient and accurate detection methods, the majority of the Android malware continue to avoid detection [6]. Moreover, Android malware threat landscape is constantly changing as malware writers continue to fix a variety of evasion techniques to avoid detection. This poses significant challenges to existing commercial anti-malware systems [7]. A useful review of Android vulnerabilities and analysis of challenges Android malware prevention is faced with is described in [8]. In order to address the challenges posed by Android malware, machine learning techniques with a single classifier have been traversed extensively. However, the performance of each classifier is different since each classifier is influenced by the induction algorithms (e.g., nave Bayes) used to produce the classifiers [9]. To address this problem, multi-tier classifier based methods have been shown to improve single classifier [9] [10]. Therefore, in Android malware rapid growth coupled with the fact that the standard anti-malware solutions only capable of detecting known instances of malware with low rates [10], there is an urgent requirement to develop effective security solution with high success rates [6] [11]. In this paper, we propose an Iterative Classifier Fusion System (ICFS) for Android malware detection. ICFS is a multi-tier framework with a minimal number of classifiers and includes a basic feature selection method into several iterative feature selection steps, where a subset of applicable features from the total number of features are selected. Our approach is motivated by the importance and the need for the depletion of volume of big data in malware detection problem [12] [13], which in turn reduces the complexity of the malware detection solutions as well as the easing of the environmental impact of their operation [14]. To the best of our knowledge, this is the first work devoted to the design of a multi-classifier system of minimal size for Android malware detection. A unique contribution of our work is summarised as follows: We present a multi-tier classification model with the minimal number of classifiers for the detection of Android malware; We develop the malware detection as a binary classification problem, we propose a general purpose meta-procedure that iteratively invokes a given base feature selection method for a given base classifier; and we present an extensive performance analysis on several base classifiers and, respectively, basic feature selection methods. The rest of the paper is categorized as follows. In Section 2, the malware detection problem formulation is presented. In Section 3, related work review is presented. In Section 4, the architecture of the ICFS and its various components are described. In Section 5, a detailed verifiable performance and analysis of

the proposed algorithm is presented. The conclusion is presented in Section 6.

II. PROBLEM OVERVIEW

In this section, the problem of malware classification is formulated as a binary classification problem. Suppose we are given a set of n programs, P = {p1,p2,...,pn}, where pi ∈ P can be a approachable or a malware. A malware is a software that has malicious intent such as data exfiltration. Each program pi ∈P is defined by a pair pi = (vi,li) (1) where vi is a feature vector and li is a class label and defined respectively as follows : vi = {a1,a2,⋯ ,az} (2) li ∈ (lm,lb,lu) (3) where aj | 1 ≤ j ≤ z represents a feature vector, lm is the malware label, lb is the approachable label and lu is the unknown. Let C = {c1,c2,...,cm} be the set of m base classifiers that are constructed to collaboratively label the input programs as either malware (lm) or benign (lb). Each classifier ci ∈ C is defined as a three tuples, ci = (Li,Ti,qi) where qi denotes the accuracy (i.e., malware detection rate),Li is a labelling function and Ti is the computational time of a classifier ci. The labelling function (Li) maps an instance v to a class label l with qi accuracy on the validation set: Li(v) : v → l (4) Our goal is to predict a label (i.e., lb or lm) for each pi ∈P | 1 ≤ i ≤ n, pi = lu Let TC and QC be the computational time and the accuracy (i.e., detection rate) of the m classifiers respectively. The overall malware detection accuracy rate and the computation time of the C classifiers are defined as follows:

$QC = \sum_{i=1}^{m} q(e_i) | c_i \in C$

$TC = \sum_{i=1}^{m} T(c_i) | c_i \in C$

The main objective of the malware detection algorithm is to classify a setP of n programs by assigning to each program a malware (lm) or benign (lb) label based on their features so as to optimize the quality of the labeling subject to some constraints. For every program pi ∈P and the available labels l, we introduce a variable xp which is equal to one if a malicious pi is correctly labelled with label lm and zero otherwise. We frame malware detection problem as a binary classification problem (i.e., one class for malware and another for non-malware) and formally define it as follows.

$\max \sum_{i=1}^{m} q(c_i).x_p \mid c_i \in c$

$S.T\, pi \in \{lm,lb\}, \square pi \in P, 1 \le i \le n, \sum_{i=1}^{m} ci \le B, \; Tc \le \phi, \sum xp = 1, \square p \in P, xp \in \{1,0\}, \square p \in P$

The objective function 7 maximizes the progressive malware detection rate (QC) of the m classifiers. The constraint in Eq.(9) states that each unlabelled program, pi ∈ P | 1 ≤ i ≤ n, must be labelled either as a agreeable or as a malware. The constraint in Eq.(10) states that the classifiers in the multi-tire classifier system must not exceed the minimum number of classifiers. The constraint in Eq. (11) states that the computational cost of the m classifiers must not exceed the threshold level (Φ). Constraint 2 shows that each instance can only be assigned one label. Determining the minimum number of classifiers (B) required to obtain the same level of performance combining large number of classifiers for the method of interest is based onintuition. Basically, we believe that a system made up of just two classifiers has no way of determining which of them is wrong in case they disagree, and so a system like this cannot be more effective than each of the original simple classifiers involved. This means that a minimal system improving on the constructiveness of simple basic classifiers has to include at least three classifiers.

III. RELATED WORK.

The rapid rise of Android malware has attracted serious interest among researchers and commercial security solution providers. The article in [15] highlights the current state of knowledge on Android malware detection methods. Existing Android malware detection and classification methods can generally be categorized as signature-based and machine learning-based techniques. In signature-based methods, malware signatures are extracted from the malware code and raw file contents. The extracted signature constitutes the principal method for identifying an instance of malware. Signature-based approach is the frequent techniques used in commercial anti-malware products. A recent study of various popular commercial anti-malware software by Zhou et al. [11] reveal that their detection rate of Android malware ranges between 20.2% and 79.6%. These studies indicate that traditional signature-based methods can be simply avoided by simple attacks such as bytecode-level transformation. Further discussion regarding why previous security solutions created for traditional computers fail to be suitable for Android devices is given in [8]. The study in [16] examines whether current antimalware techniques in Android are adaptive against Dalvik bytecode transformations. The results of this study demonstrate the need of finding detection methods robust against confuse. An advanced behavior-based method that is specifically designed to prevent system-call adminstrative vulnerability is discussed in [7]. The proposed technique applies asymptotic equipartition property to extract significant call sequences for perceive malware. In [17] an automated dynamic analysis framework is proposed for the detection of the analysis environment aware Android malware. The proposed system triggers malicious behaviors automatically by invoking restorative User-Interface events and objective broadcasts. DroidAnalyst [18] combines static and dynamic analysis. FlowMine [19] applies data flow path to differentiate malicious Android apps from benign ones. The experiments on a large dataset show that FlowMine achieved excellent results classifying correctly 96% of benign apps and 98% of malware. A static mechanism for the detection of malware that employs both permission-based and API-call based features for Android is proposed in [21]. It trains a multi-classifier system and then applies a collaborative approach depend on probability theory to combine the decisions of the classifiers our approach employs both dynamic and static analysis to classify malware. With more recent Android malware shear using very sophisticated anti-detection techniques, a better solution than the conventional signature-based is required. As a result, there has been significant research in machine learning techniques to detect Android malware. The framework developed by Deepa, et al. [?] uses three feature selection methods combined with J48 classifier. The designed system used 600 features and 600 instances of Android malware and achieved the correctness of 88.75% by combining correlation feature selection with AdaBoost and J48 classifier.An approach for cropping ensemble classifiers is proposed in [9]. MADAM [22] is a multi-level system for the detection of Android malware. LIME [23] is a large iterative multi-tier ensemble classifiers for malware detection. These studies have shown that multi-classifier systems do increase the accuracy of malware detection as compared to single-classifier systems. However, most of these systems are quite large and cannot be used for smartphones. This is because they require considerable processing and storage capacities, which subsequently rises communication overhead as well as classifier training time and

forecast time. [9]. Also, these approaches only focus on the detection accuracy, but neglect the computational cost. Malware detection approaches use some features to characterize malware and differentiate them from benign Android Apps. Thus, the first step is to generate some features that represent the input instance for the purpose of classification. Zhao, et al. [6] note that existing research on Android malware detection typically tend to manually select features. They also note that feature selection algorithms are rarely used which had led to some major limitations, and in few cases where feature selection methods used, they typically significant a specific classification algorithm to assess their performance. Unlike the exiting work, we use feature selection algorithms and we test our proposed feature selection method on a group of classification algorithms.

IV. ITERATIVE CLASSIFIER FUSION SYSTEM

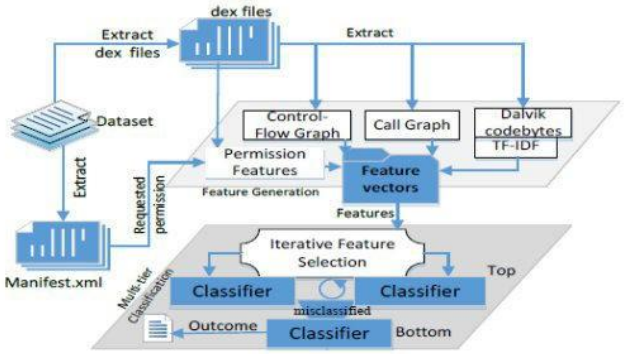

Figure.1. shows the high level architecture of the proposed Iterative Classifier Fusion System (ICFS).

The three main components of ICFS are the Feature Extractor, the Iterative Feature Selection (IFS) and the Multitier Classification features from the sample dataset.. The multi-tier classification model focuses on building models with machine learning algorithms.

2.1 Feature Generation Relevant features that can be used to uniquely characterize malware and distinguish it from benign Android Apps are required for classification purposes. Generally, these features are classified as static features or dynamic features. In our approach both static and dynamic features of Android application are removed. This is accomplished by combining several different methods. More specifically, we used APKInspector [25] to extract static features and DroidBox [26] to extract behavior based dynamic features. We note that our main aim is to design a appropriate architecture for combining a minimal number of base classifiers with feature selection method for application in further work. This is why we applied only features extracted using other ready Android analysis applications, since these features can already be used to compare different architectures and can provide further guidance for creating more advanced independent tools in the future. This can lead to several different versions of final malware detection products. The features extracted via APKInspector [25] include the authorization, Dalvikcode, and two graphs based features extract from raw bytecode: Control Flow Graph (CFG) and Call Graph. A CFG is an abstract illustration of software composed of nodes and links where the nodes represent atomic blocks (excluding jump) of instructions while the links represent the possible flow paths in the program. Android security is built on a permission-based mechanism. Specifically, applications request permission to access some controlled elements (e.g. GPS, camera) as well as sensitive components of the operating system (e.g. owners phone contacts database). The requested permissions extracted from the AndroidManifest.xml file were included in the initial large set of all features. We note that a requested permission may not be used. Thus, after we have extracted the list of requested permissions, the dex files of samples in the dataset were decompiled and used to determine which permissions were actually used by each instance. Since the Dalvik bytecode uses strings to represent constant, field, method and class, we can extract significant n-grams from sequences of opcodes in the Dalvik bytecode as features from the decompiled files. Note that n-grams have been broadly used in research and have been shown effective for solving various problems. For example, the papers [27] apply the opcode n-grams and explore the benefit of using them to detect Android malware. In order to extract significant n-grams from sequences of opcodes in the Dalvik bytecode, we applied the term frequency–inverse document frequency (TF-IDF) indices. Specifically, suppose we have a sequences of opcodes in the Dalvik bytecode are denoted as B = {b1,b2,...,bn}, which consists of n instances of malware and benign software. Furthermore, suppose we have a set of m terms, T = {t1,t2,...,tm} that is being analysed. The term frequency– inverse document frequency (F(s,n)) is defined as follows: F(s,n) = f(s,n) ×I(s,n). (14) Where f(s,n) is the frequency of a term and I(s,n) is the inverse document frequency. The frequency of a term (f(s,n)) is computed as follows: model. The Feature Generator is responsible for extracting all f(s,n) = N(s,n) ÷ i=1 $^{m}\sum$N(tj,n) Similarly, the inverse document frequency I(s,n) is computed as follows: I(s,n) = log (n ÷D(s)) where D(s) is the document frequency of the word s. The whole large collection of all extracted features was then passed on to our IFS feature selection procedure for iterative selection of features for classifiers within the architecture.

2.2 Iterative Feature Selection Iterative feature selection (IFS) is a navel procedure applied as a part of ICFS fusion system. This is a general purpose meta-procedure iteratively invoking a given base feature selection method for a given base classifier. We use the following code to define IFS: Fi ←IFS(B,S,D,ni) where B denotes the base classifier for which IFS selects a set of Fi features from D = {d1,d2,...,dn} dataset using the basic feature selection method S given as input to IFS to be invoked i number of times iteratively, where $1 \leq i \leq 3$. The parameter ni denotes the number of features selected during the ith iteration of IFS execution. Our experiments presented below show that in the special case of Android malware it does not make sense to increase the number of iterations to exceed 3. Thus we restrict the maximum number of iterations to 3, even though theoretically it can be imagined and considered for higher number of iterations. For i = 3, the IFS procedure will be invoked three times. The first execution will select |F1| = n1 features from D. The second and the third execution of IFS will select |F2| = n2 and |F3| = n3 features respectively

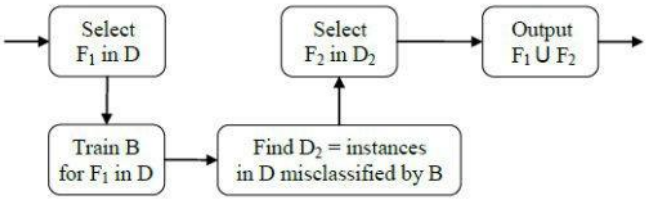

Figure.2. Iterative feature selection illustration.

Fig. 2 illustrates the IFS procedure for two iterations (i.e., i =

2). The first iteration of IFS(B,S,D,n1) uses the dataset D and applies the feature selection method S to select a set of F1 ={f1,f2,...,fn1} features with cardinality |F1| = n1 as follows:

F1 ←IFS(B,S,D,n1) Let xi = 1 denote an incorrectly classified instance of D and xi = 0 denote correctly classified instances by classifier B respectively. Therefore, all misclassified instances by B on the F1 features are collected in a new dataset D1 as follows:

D ={di,if xi = 1 and 0,otherwise}

These conditeration of IFS(B,S,D,n2) is similar to the first, with the only difference being that it selects a set of F2 = {f1,f2,...,fn2} features with cardinality |F2| = n2 from $\bar{}$ D instead of the original dataset D.

F2 ←IFS(B,S,D1,n2)

Note that this is essentially re-sampling instances on the feature selection process. We note that although there are some approaches that used feature selection methods for Android malware classification, they do not assess the effect of resampling occurence on the feature selection process [6]. The total set of features by IFS in the above example is the union of the sets selected from all elements of the dataset D at each of the two iterations as follows:

F ← F1 ∪ F2

2.3 Multi-tier classifiers The ICFS uses a multi-tier classification model for classifying Android apps as benign or malware. More specifically, ICFS uses three different base classifiers. These classifiers are organised into two tiers where the top tier contains two classifiers while the bottom tier has one classifier.

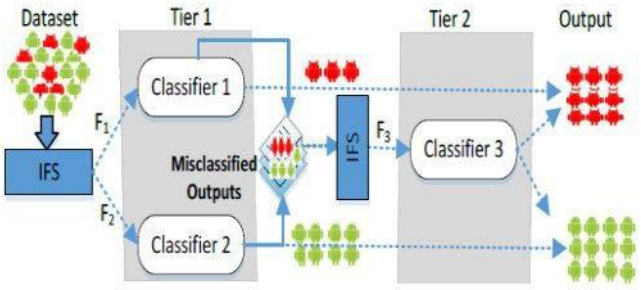

Figure.3. Block diagram of the multi-tier classification model.

Fig. 3 is a graphical illustration of the multi-tier classification model, where the arrows indicate the workflow. The three classifiers, C1, C2 and C3 are different classifiers. The stimulation for using several classifiers is to utilize the collective projective capabilities of the three classifiers to achieve greater accuracy than the individual classifier. Generally speaking, the Android apps will be classified first by the two classifiers in the first tier. The apps that are misclassified by the top tier classifiers will be fed to the classifier at the bottom tier for further analysis and classification. The training of ICFS and its operation during testing stage, or application to new data stage, are two different separate processes using disjoint datasets (cf. [28]). In the training stage, ICFS uses the training set D = {d1,d2,...,dn} to select a set of F1 = {f1,f2,...,fn1} with cardinality of n1 and another set of F2 = {f1,f2,...,fn2}features with cardinality of n2.

F1 ←IFS(C1,S,D,n1)

F2 ←IFS(C2,S,D,n2) The classifier C1 is trained to recognize malware and benign software using F1 collection of features. Likewise, classifier C2 is trained to detect malware using F2 set of features. After training, both C1 and C2 categorize each instance of D as either malware or benign software. All instances of D that were misclassified by C1 and C2 are placed into a new subset D.

D ={di, if C1 ≠ C2 0, otherwise

The third classifier C3 is trained to learn to make decisions specifically in those situations where C1 and C2 arrive to different conclusions. First, IFS extracts the features of the set F3 from all instances of D.

F3 ←IFS(C3,S,D1,n3) Then the third classifier C3 is trained to classify the instances of D as malware or benign software using features in F3. The testing stage analyzes new instances of software not considered previously, or instances of the testing set T = {t1,t2,...,tm} such that T ≠ D. In this stage, ICFS utilizes the feature sets F1, F2, F3 produced during the training stage. Specifically, IFS remove the features of the set F1 from all new instances of T. The F1 features are used by classifier C1 to recognize malware and benign software for this collection of features for each new instance. Likewise, IFS extracts the features of the set F2 from all new instances of T. Then C2 uses these features and produces an output malware or benign, too. If both C1 and C2 agree and label an instance as benign, then ICFS immediately categorizes this instance as benign. Likewise, if C1 and C2 agree and label an instance as malware, then ICFS classifies this instance as malware. All instances labeled as malware by one classifier and as benign by the other classifier are saved into a new subset T. Then IFS extracts the features of the set F3 from T instances. The classifier C3 then uses the set F3 features to make the final decision and assign the label benign or malware to document T instances.

2.4 Basic base classifier We use the notation ICFS(C1,C2,C3,S) to represent the three base classifiers C1, C2, and C3 and a basic feature selection method S used in ICFS. As building blocks for constructing ICFS system, we tested several classifiers capacity to function as the base classifiers (C1, C2, C3) under the method S. The chosen base classifiers are J48, LibSVM, MP, NBTree, NNge. We have chosen these base classifiers, because they represent important types of classification systems, are appropriate, well known, have been used in the design of effective systems in different domains by other authors, and have open source execution included in WEKA [31].

Since the performance of LibSVM depends on the kernel type used [33], we will explore the various kernel types given by the following equation:

LK(x,y) = <x,y>, polynomial kernel (PK) given by the equation

PK(x,y) = $(\gamma\langle x,y\rangle + c0)^d$, radial basis function kernel (RBFK) given by the equation

RBFK(x,y) = $e^{-(\langle y-x,y-x\rangle 2\gamma)}$, and sigmoid kernel (SK) given by the equation

SK(x,y) = tanh(γ<u,v>+c0).

2.5 Basic Feature Selection Methods In this section, we discuss the basic feature selection methods used in several layers of ICFS as a part of IFS process. Given a set of original features, the feature selection algorithm selects a subset of most relevant features based on certain criterion, which typically results in better accuracy for classification, lower computational cost, and magnified model interpretability. In this paper, we considered three different feature selection approaches: Classifier Feature Evaluator (CFE) and Information Gain Feature Evaluator (IGFE) and the Wrapper Subset Evaluator (WSE). The CFE assesses features by invoking the classifier and determining the significance of each feature for this classifier to rank all features in the order of their significance for the classifier. IGFE evaluates features

by computing the information gain each of them achieves. Specifically, given a feature X, IGFE is expressed as:
IGFE = E(X)−E(X/Y) where E(X) and E(X/Y) represent the entropy of the feature X before and after observing the feature Y respectively. The entropy of a feature X is expressed as follows:
$E(X) = -\sum_{x \in X} p(x)\log_2(p(x))$ where p(x) is the marginal probability density function for the random variable
X. Similarly, the entropy of X relative to feature Y is expressed as follows:
$E(X|Y) = -\sum_{x \in X} p(x) \sum_{x \in X} p(x|y)\log_2(p(x|y))$ where p(x|y) is the conditional probability of x given y. The higher the reduction of feature X entropy, the greater the significant of feature X is. The WSE searches for a set of features tailored for the use with any given classifier. WSE incrementally assesses subsets of features by using a classifier indicated as an input parameter to WSE. It can employ several different search strategies. We paired the Wrapper with the genetic search method. Our experiments compared the outcomes obtained by WSE invoking the following three techniques in searching instances: Best First Search (BFS), Genetic Search (GS), and Particle Swarm Optimization (PSO), which searches through subsets of the space of features using the PSO algorithm expounded in [29].

V. PERFORMANCE ANALYSIS

In this section, the performance analysis of the iterative classifier fusion system for the detection of Android malware is presented. We conducted experimental study to determine the best combinations to be deployed for C1, C2, C3 and S in the operation of ICFS (C1,C2,C3,S). We will also present the experimental setup, the performance metrics we used and the experimental results and discussions of the results.

2.6 Experimental setup We used the open source Waikato Environment for Knowledge Analysis (WEKA) toolkit [31] to implement and study ICFS. We used the MalGenome dataset of Android malware described in [11]. The dataset contains instances of Android malware divided into 49 different types of malware. It was adjunct by a small number of more recent malware samples collected at the Contagio Mobile Malware Mini Dump [32] that makes mobile malware samples available. In the experiments, we used IFS with 1, 2 and 3 iterations. We substituted the values n1 = 20, n2 = 10, and n3
= 5 in IFS for these iterations, since these values are proportionate with the numbers of features considered as effective options in previous research.

2.6.1 Performance metric We used the Area Under Curve (AUC) as the performance measure to assess the effectiveness of the proposed iterative classifier fusion system for the detection of Android malware. AUC is commonly used as a performance metric Also, it is much more suitable to have a single metric to compare the different models. AUC is defined as an area under the ROC curve representing, for a series of cut-off values, the dependence of true positive rates on the false positive rates.

The ROC curve can also be defined in terms of sensitivity and specificity:
Sensitivity =TP/ (TP + FN Specificity =TN/ (TN + FP) where TP is the true positive rate, TN is the true negative rate, FN is the false negative instances, and FP is the false positive illustration. The ROC curve displays the dependence of sensitivity and specificity for each of the cut-off values. The AUC takes on values belonging to the interval from 0.5 to 1. The greater the AUC value, the better malware detection is achieved by the system. In particular, if AUC value equal to 1 means that perfect results have been obtained. To prevent overstating in evaluating the effectiveness of classification systems, during all our experiments we used the standard and well known technique of tenfold cross validation. It means that the dataset D is divided into a disjoint union
D = D1 ˙ ∪ D2 ˙ ∪ D3 ˙ ∪ ... ˙ ∪ D10. often stratified subsets or folds D1,D2,...,D10.These subsets are then used to conduct ten experiments as follows. In the first experiment the union of sets D2,D3,...,D10 is applied to train the classification system and is called a training set, and the set D1 is then employed for testing the effectiveness of the trained system and is called a testing set. The AUC achieved during testing on the set D1 is recorded. Likewise, for i = 2,...,10, in the i-th experiment, the set Di is used as a testing set, whereas the union of all other sets,
Ti = D1 ˙ ∪ ... ˙ ∪ Di−1 ˙ ∪ Di+1 ˙ ∪ ... ˙ ∪ D10 is used as a training set. The classification system is trained on Ti and then its effectiveness is determined for the disjoint subset Di. An average of the AUC outcomes obtained and recorded during these ten experiments is then calculated and is output as the final outcome of the tenfold cross validation procedure.

2.6.2 Results and Discussion In the experiments, we investigate the effectiveness of all possible options for substituting a triplet of basic classifiers and the basic feature selection method to perform the roles of classifiers C1, C2, C3 and S in ICFS. We refer to the case where a triplet of classifiers are selected from J48, MP, NB Tree, NNge to perform the roles of C1, C2, C3 as ”ICFS base triplets”. In contrast, we refer to the case where all possible pairs of the classifiers J48, MP, NB Tree, NNge are combined with LibSVM to perform the roles of C1, C2, C3 as ”ICFS LibSVM triplets”. In the rows of the diagrams with outcomes of experiments, we list all possible triplets X, Y, Z of the base classifiers to be substituted for C1, C2 and C3 during experiments, one triplet per row of a diagram, subject to the following simplifications .Any particular triplet X, Y, Z of classifiers produces exactly the same result in ICFS(X,Y,Z) as the alternative triplet Y, X, Z in ICFS(Y,X,Z), because C1 and C2 perform identical roles. This is why we include only one of these triplets in the diagrams with outcomes of experiments. Each diagram contains outcomes obtained by the corresponding procedure for 1, 2 and 3 iterations of the IFS. Performance of ICFS base triplets In this section, we look at performance results of ICFS base triplets in which the J48, MP, NB Tree, and N Nge base classifiers form the triplets. The following five diagrams (Fig. 4 to Fig. 8) show the outcomes achieved by all different triplets of the classifiers J48, MP, NB Tree, NNge when CFE, IGFE and WSE feature selection methods are used as a basic feature selection in IFS

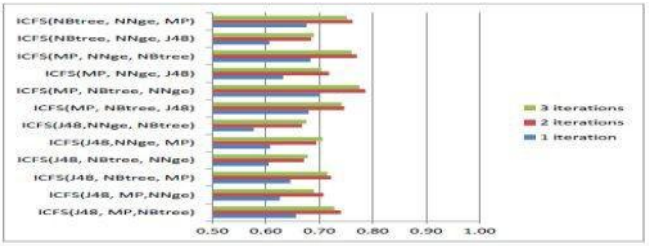

Figure.4. shows the performance of the ICFS base triplets using Classifier Feature Evaluator (CFE)

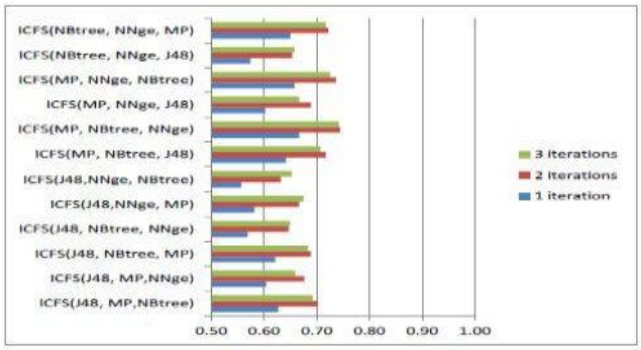

Figure.5. depicts the performance outcome of the ICFS base triplets using Information Gain Feature Evaluator (IGFE) feature

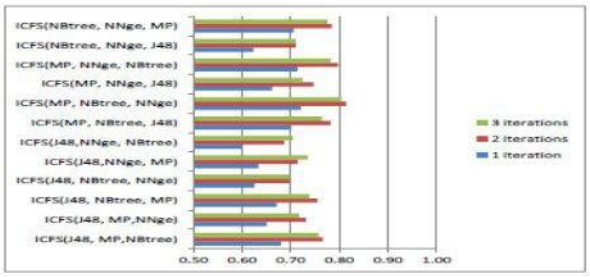

Figure.6. Performance of ICFS base triplets using WSE with BFS as basic feature selection in IFS

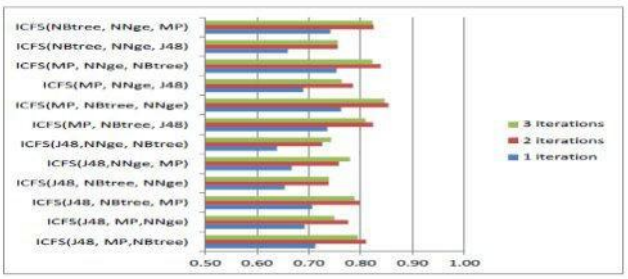

Figure.7. Performance of ICFS base triplets using WSE with GS as basic feature selection in IFS

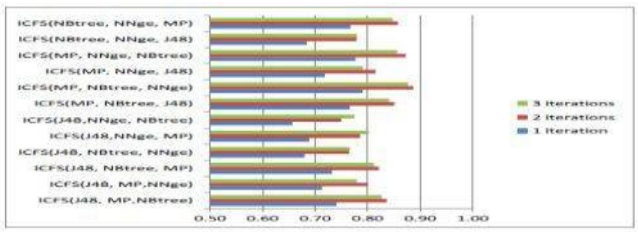

Figure.8. performance of ICFS base triplets using WSE with PSO as basic feature selection in IFS

Fig. 6 to Fig. 8 show the performance of the ICFS base triplets using Wrapper Subset Evaluator (WSE) feature selection methods as a basic feature selection in IFS Each classifier used the resulting ranking and chose the required number of the most significant features. For WSE-based feature selection algorithm,. Thus the best outcomes were obtained by the ICFS base triplets employing the WSE feature selection with PSO search strategy.

Performance of ICFS LibSVM triplets In this section, we present performance results of ICFS LibSVM triplets formed from all possible pairs of the classifiers J48, MP, NB Tree, NNge combined with Lib SVM .In the experiments , the feature selection method used as a basic feature selection in IFS is fixed to WSE with PSO. The following four diagrams

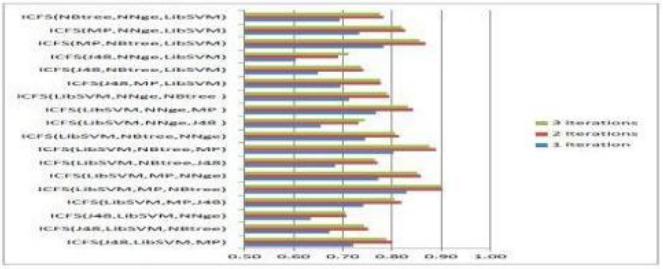

Figure.9. performance of ICFS LibSVM +LK

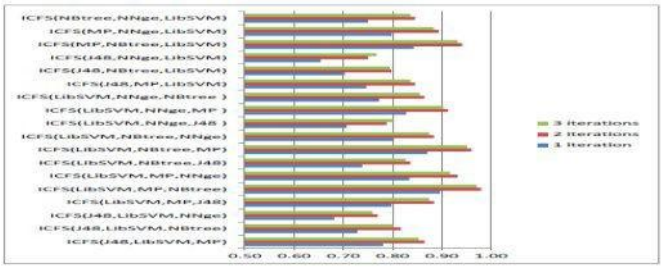

Figure.10. performance of ICFS LibSVM +PK triplet

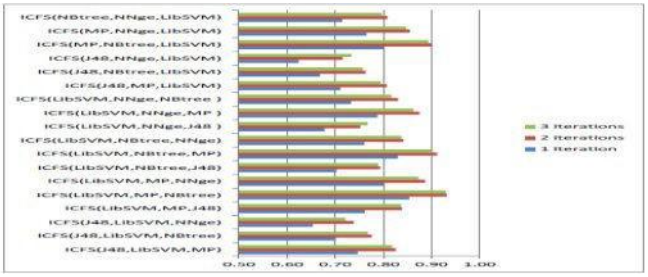

Figure.11. performance of ICFS RBFK triplet

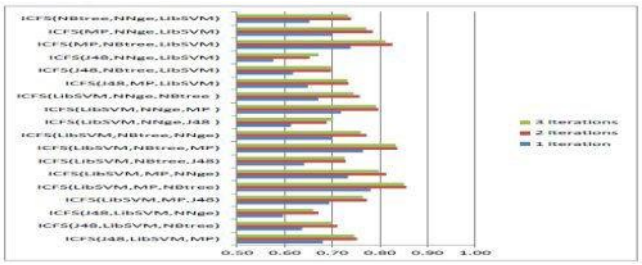

Figure.12. performance of ICFS LibSVM +SK triplet

J48, MP, NB Tree, NNge combined with LibSVM. The top six results show the performance of LibSVM as C3 while varying C1 and C2 between the J48, MP, NB Tree, and NNge classifiers. The next nine results show the performance of LibSVM as C1 while varying C2 and C3 between the J48, MP, NB Tree, and NNge classifiers. The last three results set C1 = J48 and C2 = LibSV M while C3 varies between MP, NB Tree, and NNge classifiers. In summary, the results of the experiments show that LibSVM with polynomial kernel (PK) achieved the best results in ICFS. Thus, the best optionforICFSinFig.4 toFig.12istheICFS(LibSVM, MP, NB tree, WSE+PSO) using WSE with PSO as basic feature selection in IFS.

VI. CONCLUSION

The aim of this paper is to develop the most effective classification system with the minimal number of classifiers for the detection of Android malware. To this end, we introduced a novel system ICFS combining the smallest possible number of basic classifiers in fusion with an iterative feature selection to improve their performance. The system invokes the classifiers iteratively combining them with new

iterative feature selection procedure. This paper uses the wellknown MalGenome dataset to carry out a comprehensive empirical study comparing the effectiveness of various options that can be employed in ICFS and also comparing the effectiveness of ICFS with several traditional systems. The results of experiments presented in the paper show that ICFS achieved higher AUC in comparison with several other traditional classifiers. The best outcomes have been obtained by the ICFS using LibSVM with PK, MP, NBtree, and IFS based on WSE with PSO.

VII. ACKNOWLEDGMENTS

This paper will not have been possible without the help of Maliha Omar and we sincerely appreciated the help. The authors are grateful to three reviewers for comments that have helped to improve this article

VIII. REFERENCES

[1]. C.J. DOrazioa, and K-K. R. Choo,"Circumventing iOS security mechanisms for APT forensic investigations: A security taxonomy for cloud apps," Future Generation Computer Systems, Available online 17 November 2016, http://dx.doi.org/10.1016/j.future.2016.11.010

[2]. Q. Do, B. Martini, K-K. R. Choo,"A Data Exfiltration and Remote Exploitation Attack on Consumer 3D Printers", Information Forensics and Security IEEE Transactions on, vol. 11, pp. 2174-2186, 2016, ISSN 1556-6013.

[3]. K.-K. R. Choo, "The cyber threat landscape: Challenges and future research directions," Computers & Security, vol. 30, pp. 719–731, 2011.

[4]. A. Chonka, and J. Abawajy Detecting and mitigating HX-DoS attacks against cloud web services, IEEE 15th International Conference on Network-Based Information Systems (NBiS), Pages 429-434, 2012.

[5]. S. Rajan, W. van Ginkel, N. Sundaresan, A. Bardhan, Y. Chen, A. Fuchs, A. Kapre, A. Lane, R. Lu, P. Manadhata, J. Molina, A. C. Mora, P. Murthy, A. Roy, S. Sathyadevan, and N. Shah, Expanded top ten big data security and privacy challenges, Cloud Security Alliance, available at https://downloads. cloudsecurityalliance. org/initiatives/bdwg /Expanded Top Ten Big Data Security and Privacy Challenges.pdf, viewed on 19 September 2016.

[6]. K.Zhao,D.Zhang,X.Su,andW.Li,"Fest:A feature extraction and selection tool for Android malware detection," in 20th IEEE Symposium on Computers and Communication, ISCC 2015, 2015, pp. 714– 720.