DroidFusion: A Novel Multilevel Classifier Fusion Approach for Android Malware Detection

Suleiman Y. Yerima, *Member, IEEE,* and Sakir Sezer, *Member, IEEE,*

***Abstract*—Android malware has continued to grow in volume and complexity posing significant threats to the security of mobile devices and the services they enable. This has prompted increasing interest in employing machine learning to improve Android malware detection. In this paper we present a novel classifier fusion approach based on a multilevel architecture that enables effective combination of machine learning algorithms for improved accuracy. The framework (called DroidFusion), generates a model by training base classifiers at a lower level and then applies a set of ranking-based algorithms on their predictive accuracies at the higher level in order to derive a final classifier. The induced multilevel DroidFusion model can then be utilized as an improved accuracy predictor for Android malware detection. We present experimental results on four separate datasets to demonstrate the effectiveness of our proposed approach. Furthermore, we demonstrate that the DroidFusion method can also effectively enable the fusion of ensemble learning algorithms for improved accuracy. Finally, we show that the prediction accuracy of DroidFusion, despite only utilizing a computational approach in the higher level, can outperform Stacked Generalization, a well-known classifier fusion method that employs a meta-classifier approach in its higher level.**

***Index Terms*—Android Malware Detection, Mobile Security, Machine Learning, Classifier Fusion, Ensemble Learning, Stacked Generalization.**

I. Introduction

In recent years, Android has become the leading mobile operating system with a substantially higher percentage of the global market share. Over 1 billion Android devices have been sold with an estimated 65 billion app downloads from Google Play alone [1]. The growth in popularity of Android and the proliferation of third party app markets has also made it a popular target for malware. Last year, McAfee reported that there were more than 12 million Android malware samples with nearly 2.5 million new samples discovered every year [2]. Android malware can be embedded in a variety of applications such as banking apps, gaming apps, lifestyle apps, educational apps, etc. These malware-infected apps can then compromise security and privacy by allowing unauthorized access to privacy- sensitive information, rooting devices, turning devices into remotely controlled bots, etc.

Zero-day Android malware have the ability to evade traditional signature-based defences. Hence, there is an urgent need to develop more effective detection methods. Recently, machine learning based methods are increasingly being applied to Android malware detection. However, classifier fusion approaches have not been extensively explored as they have been in other domains like network intrusion detection.

In this paper, we present and investigate a novel classifier fusion approach that utilizes a multilevel architecture to increase the predictive power of machine learning algorithms. The framework, called DroidFusion, is designed to induce a classification model for Android malware detection by training a number of base classifiers at the lower level. A set of ranking-based algorithms are then utilized to derive combination schemes at the higher level, one of which is selected to build a final model. The framework is capable of leveraging not only traditional singular learning algorithms like Decision Trees or Naive Bayes, but also ensemble learning algorithms like Random Forest, Random Subspace, Boosting etc. for improved classification accuracy.

In order to demonstrate the effectiveness of the DroidFusion approach, we performed extensive experiments on four datasets derived from extracting features from two publicly available and widely used malware samples collection (i.e. Android Malgenome project [3] and DREBIN [4]) and a collection of samples provided by Intel Security (Formerly, McAfee). The unique contributions of this paper can be summarized as follows:

- We propose a novel general-purpose classifier fusion approach (DroidFusion) and present its evaluation on four different datasets. DroidFusion can be applied to not only traditional learners but also ensemble learners.
- We propose four ranking-based algorithms that enable classifier fusion within the DroidFusion framework. The algorithms are utilized in building a final improved classification model for Android malware detection.
- We present the results of extensive experiments to demonstrate the effectiveness of our proposed approach. The results of experiments with singular classifiers and ensemble classifiers are presented.
- Furthermore, we present results of a performance comparison of DroidFusion with Stacked Generalization (or Stacking), a well- known classifier fusion method that is also based on a multilevel architecture.
- Datasets that we created from the feature extraction process with DREBIN and Malgenome project malware samples are released in the supplementary material.

The rest of the paper is structured as follows. Section II discusses related work while section III presents the DroidFusion framework. The investigation methodology is presented in section IV, while section V presents results, with analyses and discussions. Finally, the conclusion is given in section VI.

S. Y. Yerima is with the Cyber Technology Institute, School of Computer Science and Informatics, De Montfort University, Leicester, England. (e-mail: syerima@dmu.ac.uk).

S. Sezer is with the Centre for Secure Information Technologies (CSIT), Queen's University Belfast, Northern Ireland, UK (e-mail: s.sezer@qub.ac.uk).

Manuscript received June 03, 2017; revised September 11, 2017, accepted November 11 2017.

II. Related work

In this section we review related work on machine learning based Android malware detection. Static and/or dynamic analysis is used to extract model training features, and both methods have pros and cons. Static analysis is prone to obfuscation [5], but is generally faster and less resource intensive than dynamic analysis. Dynamic analysis is resistant to obfuscation but can be hampered by anti-virtualization [6]–[9] and code coverage limitations [10], [34].

A. Static analysis with traditional classifiers

Recent Android malware detection work that employ machine learning with static features include the following. DroidMat [11] proposed applying k-means and k-NN algorithms based on static features from permissions, intents and API (application program interface) calls, to classify apps as benign or malware. Arp et al. [4], proposed SVM based on permissions, API calls, network access, etc. for lightweight on-device detection. Yerima, et al. [12], [14] proposed an Eigenpsace analysis approach, as well as Random Forest ensemble learning models. The machine learning based detection proposed in the papers were based on API calls, intents, permissions and embedded commands. Varsha et al. [15] investigated SVM, Random Forest and Rotation Forests on three datasets; their detection method employed static features extracted from the manifest and application executable files.

Sharma and Dash [16] utilized API calls and permissions to build Naive Bayes and k-NN based detection systems. In [17], API classes were used with Random Forest, J48 and SVM classifiers. Wang et al. [18] evaluated the usefulness of risky permissions for malware detection using SVM, Decision Trees and Random Forest. DAPASA [19] focused on detecting malware piggybacked onto benign apps by utilizing sensitive subgraphs to construct five features depicting invocation patterns. The features are fed into machine learning algorithms i.e. Random Forest, Decision Tree, k- NN and PART, with Random Forest yielding the best detection performance. Cen et al. [20] proposed a detection method based on API calls from decompiled code and permissions. Their proposed method applies a probabilistic discriminative model based on regularized logistic regression (RLR). RLR is compared to SVM, Decision Tree, k-NN, Naive Bayes with information priors and Hierarchical mixture of Naive Bayes.

Wang et al. [52] applied Logistic regression, Linear SVM, Decision Tree and Random forest with static analysis for the detection of malicious apps. They utilized app-specific static features and platform-specific static features for training the machine learning algorithms. The authors reported a maximum true positive rate of 96% and false positive rate of 0.06% with the Logistic Regression classifier based on experiments conducted on 18,363 malware apps and 217,619 benign apps.

Other research papers that have investigated static features with machine learning for Android malware detection include [21]–[23], [45], [47], [48] and [54].

B. Dynamic & hybrid analysis with traditional classifiers

Some of the detection methods utilized dynamic features with machine learning, for example AntiMalDroid [24]. AntiMalDroid is a dynamic analysis behavior based malware detection framework that uses logged behavior sequence as features with SVM. DroidDolphin [25], also employed SVM with dynamically obtained features. Afonso et al. [26] utilized dynamic API calls and system call traces and investigated SVM, J48, IBk (an instance based classifier), BayesNet K2, BayesNet TAN, Random Forest and Naive Bayes. Alzaylaee et al. [27] investigated SVM, Naive Bayes, PART, Random Forest, J48, MLP (multi-layer perceptron), and Simple logistic by comparing their performances on real phones vs. emulators using dynamically obtained features. Ni et al. [46], proposed a real-time malicious behavior detection system that records API calls, permission uses, and other real-time features such as user operations. In their paper, they used SVM and Naive Bayes algorithms for detection with these run-time features.

Mahindru and Singh [53] extracted 123 dynamic permissions from 11000 Android applications which were subsequently applied to several individual machine learning classifiers including Naive Bayes, Decision Tree, Random Forest, Simple Logistic and k-star. In their experiments Simple Logistic was found to perform marginally better than the others but the malware classification accuracy of Random Forest, Decision Tree (J48) and Simple logistics were comparable.

Other works such as MARVIN [28], adopt a hybrid static and dynamic feature based approach with machine learning (SVM and L2 regularized linear classifer). MARVIN assesses the risk associated with unknown Android apps in the form of a malice score ranging from 0 to 10. Similarly, Su et al. [49] adopted a hybrid static and dynamic feature approach by performing experiments on 1200 (900 clean and 300 malware) samples. Several machine learning algorithms were investigated including Bayes Net, Naive Bayes, K-NN, J48, and SVM. The best overall accuracy of 91.1% was attained with SVM.

C. Android malware detection with classifier fusion

Previous works in intrusion detection systems such as [29]–[32] investigated classifier fusion for improving detection accuracy. This method is also being applied to the detection of Android malware. For example Milosevic et al. [50] investigated classifer fusion approach with static analysis based on Android permissions and source code-based analysis. They used SVM, C.45, Decision Trees, Random Tree, Random Forests, JRip and linear regression classifiers. The authors experimented with ensembles that contained odd combinations of three and five classifiers using the majority voting fusion method. The best fusion model achieved an accuracy rate of 95.6% using the source-code based features. However, the number of samples used in the experiments were limited (387 samples for the permissions-based experiments and 368 for source code-based analysis)

Yerima et al. [13] compared several classifier fusion methods i.e. Majority vote, Product of probabilities, Maximum probability, and Average of probabilities using J48, Naive Bayes, PART, RIDOR, and Simple Logistic classfiers. The classifiers were trained with static features extracted from 6,863 app samples, and in the experiments presented, the fused models performed better than the single classifiers.

TABLE I: Overview of some of the papers that apply classfier fusion for Android malware detection. NB = Naive Bayes; SL= Simple Logistic; LR= Linear Regression; DT = Decision Tree; VP= Voted Perceptron. AveP = average of probabilities; ProdP = product of probabilities; MaxP = maximum probability.

Paper/Year	ML algorithms	Fusion approach	# samples
Yerima et. al [13] (2014)	SVM, J48, PART, Ridor, NB, SL	Majority vote, ProdP, AveP, MaxP	6,863
Coronado-de-Alba et. al [33] (2016)	Random Forest, Random Committee	Meta-ensembling Random Forest in Random Comm.	3,062
Milosevic et. al [50] (2017)	SVM, C.45, RT DT, JRip, LR, Random Forest	Majority vote	387 368
Wang et. al [51] (2017)	SVM, KNN, NB, CART, Random Forest	Majority vote	116,028
Idrees et al. [55] (2017)	MLP, DT, Decision Table	Majority vote, AveP, ProdP	1,745
DroidFusion (This paper)	RT, J48, RepTree, VP, Random Forest, Random Comm., Random Sub., AdaBoost	Multilevel weighted Ranking-based approach	3,799 15,036 36,183

Wang et al. [51] extracted 11 types of static features and employ multiple classifiers in a majority vote fusion approach. The classifiers include SVM, K-Nearest Neighbour, Naive Bayes, Classification and Regression Tree (CART) and Random Forest. Their experiments on 116,028 app samples showed more robustness with the majority voting ensemble than with the individual base classifiers.

Idrees et al. [55] utilize permissions and intents as features to train machine learning models and applied classifier fusion for improved perfromance. Their experiments were performed on 1745 app samples starting with a performance comparison between MLP, Decision Table, Decision Tree, Random Forest, Naive Bayes and Sequential Minimal Optimization classifiers. The Decision Table, MLP, and Decision Tree classfiers were then combined using three schemes: average of probabilities, product of probabilities and majority voting. Coronado-de-Alba et al. [33] proposed and investigated a classifier fusion method based on Random Forest and Random Committee ensemble classifiers. Their approach embeds Random Forest within Random Commitee to produce a meta-ensemble model. The meta-model outperformed the individual classifiers in experiments performed with 1531 malware and 1531 benign samples. Table I summarizes papers that have investigated classifier fusion for Android malware detection.

In contrast to all of the existing Android malware detection works, this paper proposes a novel classifier fusion approach that utilizes four ranking based algorithms within a multilevel framework (DroidFusion). We evaluated DroidFusion extensively and compared its performance to Stacking and other classifer fusion methods. Next, we present DroidFusion.

III. DroidFusion: general purpose framework for classifier fusion

The DroidFusion framework consists of a multilevel architecture for classifier fusion. It is designed as a general purpose classifier fusion system, so that it can be applied to both traditional singular classifiers and ensemble classifiers (which themselves employ a base classifier usually to produce different randomly induced models that are subsequently combined). At the lower level, the (DroidFusion) base classifiers are trained on a training set using a stratified N-fold cross validation technique to estimate their relative predictive accuracies. The outcomes are utilized by four different ranking-based algorithms (in the higher layer) that define certain criteria for the selection and subsequent combination of a subset (or all) of the applicable base classifiers. The outcomes of the ranking algorithms are combined in pairs in order to find the $strongest\ pair$, which is subsequently used to build the final DroidFusion model (after testing against an unweighted parallel combination of the base classifiers).

A. DroidFusion model construction

The model building i.e. training process is distinct from the prediction or testing phase, as the former utilizes a training-validation set to build a multilevel ensemble classifier which is then evaluated on a separate test set in the latter phase. Figure 1, illustrates the 2-level architecture of DroidFusion. It shows the training paths (solid arrows) and the testing/prediction path (dashed arrows). First, at the lower level each base classifier undergoes an N-fold cross validation based estimate of class performance accuracies. Let the N-fold cross validated predictive accuracies for K base classifiers be expressed by P_{base}, a K-tuple of the class accuracies of the K base classifiers:

$$P_{base} = \{[P_{1m}, P_{1b}], [P_{2m}, P_{2b}], ..., [P_{Km}, P_{Kb}]\} \quad (1)$$

The elements of P_{base} are applied to the ranking based algorithms AAB, CDB, RAPC and RACD described later in section III-B. Let X be the total number of instances with M malware and B benign instances, where the M instances possess a label L=1 denoting malware and the B instances from X possess a label L=0 denoting benign. All X instances are also represented by feature vectors with f binary representations, where f is the number of features extracted from the given app. The features in the vectors take on 0 or 1 representing the absence or presence the given feature. Additionally, after the N-fold cross validation process (as shown in Fig. 1), a set of K-tuple class predictions are derived for every instance x, given by:

$$V(x) = \{v_1, v_2, ..., v_k\}, \quad \forall k \in \{1, ..., K\} \quad (2)$$

Note that v_1,v_2,...,v_k could be crisp predictions or probability estimates from the base classifiers. Adding the original (known) class label, l, we obtain:

$$\dot{V}(x) = \{v_1, v_2, ..., v_k, l\}, \quad \forall k \in \{1, ..., K\}, l \in \{0, 1\} \quad (3)$$

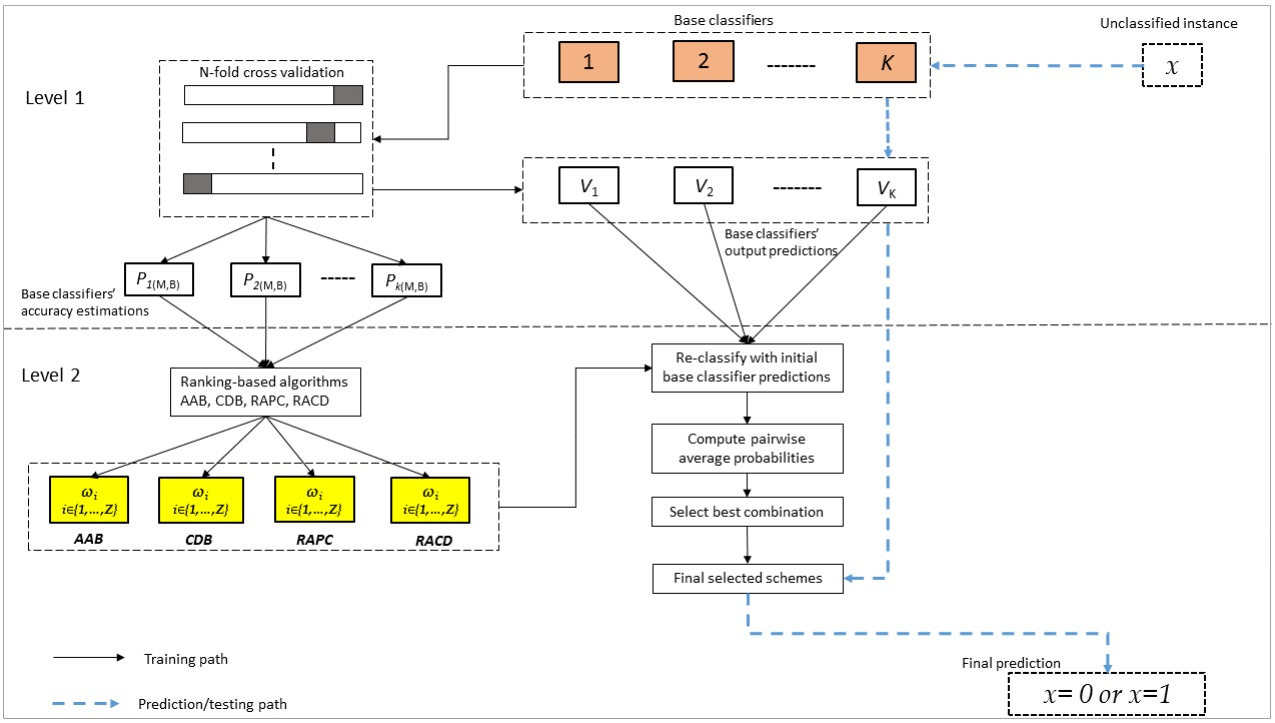

Fig. 1: DroidFusion 2-layer model architecture.

P_{base} and $\dot{V}(x)$, $\forall\ x \in X$ will be utilized in the level-2 computation during the DroidFusion model construction. Let us denote the set of four ranking based schemes by $S = \{S1, S2, S3, S4\}$. The pairwise combinations of the elements of S will result in 6 possibilities:

$$\phi = \{S1S2, S1S3, S1S4, S2S3, S2S4, S3S4\} \tag{4}$$

Our goal is to select the best pair of ranking-based schemes from S, and if its performance exceeds that of an unweighted combination of the original base classifiers, it would be selected to construct the final DroidFusion model. In the event that the unweighted combination performance is greater, DroidFusion will be configured to apply a majority vote (or average of probabilities) of the base classifiers in the final constructed model. In order to estimate the accuracy performance of each scheme in S or each pairwise combination in set ϕ, a re- classification of the X instances (in the training-validation) set is performed for each scheme or pair of schemes. The re-classification is accomplished using $\dot{V}(x), x \in X$ based on the criteria defined by the schemes in S using P_{base}. Each scheme in S derives a set of Z weights that will be applied with $\dot{V}(x)$, $x \in X$ for every instance during the re-classification process.

Let $\omega_i, i \in \{1, ..., Z\}, Z \leq K$ be the set of weights derived for a particular scheme in S. Then, to reclassify an instance x according to the schemes criterion, its class prediction will be given by:

$$C_{Sj}(x) = \begin{cases} 1 : \text{if } \frac{\sum_{i=1}^{Z} \omega_i v_i}{\sum_{i=1}^{Z} \omega_i} \geq 0.5 \\ 0 : \quad \text{otherwise} \end{cases} \qquad \forall j \in \{1, 2, 3, 4\} \tag{5}$$

Hence, the benign class accuracy performance for the given scheme is calculated from:

$$P_{Sj}^{ben} = \frac{\sum_{x=1}^{X} (C_{Sj}(x) + 1) \quad | \quad C_{Sj}(x) = 0, l(x) = 0}{B} \tag{6}$$

Where B is the number of benign instances, while the malware accuracy performance is calculated from:

$$P_{Sj}^{mal} = \frac{\sum_{x=1}^{X} C_{Sj}(x) \quad | \quad C_{Sj}(x) = 1, l(x) = 1}{X - B} \tag{7}$$

Thus the average performance accuracy is simply:

$$\dot{P}_{Sj} = \frac{B \cdot P_{Sj}^{ben} + (X - B) \cdot P_{Sj}^{mal}}{X} \tag{8}$$

Likewise, to determine the performance of each pairwise combination in ϕ:

Let $\omega_i, i \in \{1, ..., Z\}, Z \leq K$ be the first set of weights derived for the first scheme in the pair, and let $\mu_i, i \in \{1, ..., Z\}, Z \leq K$ be those derived for the second scheme in the pair. Then, to reclassify the X instances in the training-validation set according to the combination pair, the class prediction of each instance x will be given by:

$$C_{SjSn}(x) = \begin{cases} 1 : \text{if } \frac{\sum_{i=1}^{Z} \omega_i v_i + \sum_{i=1}^{Z} \mu_i v_i}{\sum_{i=1}^{Z} \omega_i + \sum_{i=1}^{Z} \mu_i} \geq 0.5 \\ 0 : \quad \text{otherwise} \\ \forall j \in \{1, 2, 3, 4\}, \forall n \in \{1, 2, 3, 4\}, \\ j \neq n, SjSn \equiv SnSj \end{cases} \tag{9}$$

Therefore, computing benign class accuracy and malware class accuracy will utilize:

$$P_{SjSn}^{ben} = \frac{\sum_{x=1}^{X}(C_{SjSn}(x)+1) \quad | \; C_{SjSn}(x)=0, l(x)=0}{B} \tag{10}$$

and

$$P_{SjSn}^{mal} = \frac{\sum_{x=1}^{X} C_{SjSn}(x) \quad | \; C_{SjSn}(x)=1, l(x)=1}{X-B} \tag{11}$$

respectively. The average performance accuracy for the pairwise schemes will then be given by:

$$\dot{P}_{SjSn} = \frac{B \cdot P_{SjSn}^{ben} + (X-B) \cdot P_{SjSn}^{mal}}{X} \tag{12}$$

$\forall j \in \{1,2,3,4\}, \forall n \in \{1,2,3,4\}, j \neq n, SjSn \equiv SnSj$. Equivalently, the unweighted majority vote class predictions for instance x is given by:

$$C_{mv}(x) = \begin{cases} 1: \text{if} \quad \frac{\sum_{k=1}^{K} v_i}{K} \geq 0.5 \\ 0: \quad \text{otherwise} \end{cases} \qquad \forall k \in \{1,...,K\} \tag{13}$$

Hence, the benign class accuracy performance for the unweighted scheme will be given by:

$$P_{mv}^{ben} = \frac{\sum_{x=1}^{X}(C_{mv}(x)+1) \quad | \; C_{mv}(x)=0, l(x)=0}{B} \tag{14}$$

Likewise, the malware class accuracy performance for the unweighted scheme is given by:

$$P_{mv}^{mal} = \frac{\sum_{x=1}^{X} C_{mv}(x) \quad | \; C_{mv}(x)=1, l(x)=1}{X-B} \tag{15}$$

Finally, the average accuracy performance for the unweighted scheme is given by:

$$\dot{P}_{mv} = \frac{B \cdot P_{mv}^{ben} + (X-B) \cdot P_{mv}^{mal}}{X} \tag{16}$$

After all the re-classifications are completed, and the average accuracies computed, the applicable scheme that will be utilized to construct the DroidFusion model is selected thus:

$$\begin{cases} arg_{\phi} max(\dot{P}_{\phi}), \\ \phi = \{S1S2, S1S3, S1S4, S2S3, S2S4, S3S4, mv\} \end{cases} \tag{17}$$

Suppose that $S1$ and $S3$ pair are selected by the operation in Eq. (17), then the class of a given unlabeled instance during the testing or unknown instances prediction (during model deployment) will be computed by Equation (9) with j=1 and n=3. Next, we describe the four ranking-based algorithms underpinning the schemes in set S that utilize P_{base} to accomplish all of the above described DroidFusion level-2 steps.

B. Proposed ranking based algorithms

The design of our proposed algorithms is influenced by the observation that most typical classifiers perform differently for both classes. That is, class accuracy performance for benign and malware are very rarely equal in magnitude. The proposed ranking based algorithms include:

- An average accuracy based ranking scheme (AAB).
- A class differential based ranking scheme (CDB).
- A ranked aggregate of per class performance based scheme (RAPC).
- A ranked aggregate of average accuracy and class differential based scheme (RACD).

1) The Average Accuracy Based (AAB) ranking scheme : With the AAB method, the ranking is designed to be directly proportional to the average prediction accuracies across the classes. In this case, base classifiers with larger overall accuracy performance will rank higher. AAB doesnt take into account how well a base classifier performs for a particular class. Let AAB be the first scheme $S1$, from set S. The algorithm is summarized as follows:

Let P_{base} be the set of performance accuracies $P_{k,c} \in P_{base}$ of K base classifiers. If m denotes malware and b, benign then the average accuracy of the k^{th} base classifier is given by:

$$a_k = 0.5 \times \sum\nolimits_{c=m,b} P_{k,c} \;\; | \;\; k \in \{1,...,K\}, \;\; 0 < P_{k,c} \leq 1 \tag{18}$$

Let $A \leftarrow a_k, \forall k \in \{1,...,K\}$ be a set of the average predictive accuracies, to which a ranking function $Rank_{desc}(.)$ is applied:

$$\bar{A} \leftarrow Rank_{desc}(A) \tag{19}$$

Thus, $\bar{A}$ contains an ordered ranking of the Level-1 base classifiers average predictive accuracies in descending order. Next, the top Z rankings are utilized in weight assignments as follows:

$$\omega_1 = Z, \omega_2 = Z-1, ..., \omega_Z = 1, \qquad Z \leq K \tag{20}$$

Thus, the AAB class prediction $C(x)$ for instance x in the training-validation set is given by Eq. (5) or given by Eq. (9) when used in the pairwise combination with another scheme.

2) The Class Differential Based (CDB) ranking scheme: With the CDB method, the ranking is directly proportional to the average predictive accuracy and inversely proportional to the absolute value of the performance difference between the classes. Assuming a binary classification problem, this approach will be less likely to favour the decision from a base classifier that exhibits much higher accuracy in one class over the other but will assign larger weights to $good \; classifiers$ that perform relatively well in $both \; classes$. The CDB procedure is described as follows:

Suppose the CDB method is taken as scheme $S2$, let the average accuracy of each base classifier be given by a_k in equation (18) and define $\bar{D}$ with cardinality K as a set of ordered rankings in descending order of magnitude. Calculate d_k proportional to average accuracies and inversely proportional to absolute difference of inter-class accuracies:

$$d_k = \frac{a_k}{|P_{k,m} - P_{k,b}|} \;\; , \;\; k \in \{1,...,K\} \tag{21}$$

Let $D \leftarrow d_k, \forall k \in \{1,...,K\}$ be a set of the d_k values, to which the ranking function $Rankdesc(.)$ is applied:

$$\bar{D} \leftarrow Rank_{desc}(D) \tag{22}$$

With $\bar{D}$ containing the ordered rankings of d_k values, the top Z rankings are also utilized to assigned weights according to

Eq. (20). Thus, the $S2$=CDB class prediction for an instance x is determined from Eq. (5). Whenever $S2$=CDB is used (in conjunction with another scheme) within a pair in the set expressed by Eq. (4), then equation Eq. (9) will be used for the class prediction of the instance.

3) The Ranked Aggregate of Per Class accuracies (RAPC) based scheme: In the RAPC method, the ranking is directly proportional to the sum of the initial per class rankings of the accuracies of the base classifiers. This method is more likely to assign a larger weight to a base classifier that performs very well in both classes. RAPC is summarized as follows.

With $\bar{F}$ defined as the set of ordered rankings with cardinality K, given the initial performance accuracies of $P_{k,c}$ of the K base classifiers :

$$\begin{cases} P_m \leftarrow P_{k,c} \quad where \;\; c \neq b \\ P_b \leftarrow P_{k,c} \quad where \;\; c \neq m \end{cases} , k \in \{1, ..., K\}, c \in \{m, b\} \tag{23}$$

We then apply the ranking function $Rank_{desc}(.)$ to both:

$$\begin{cases} \bar{P}_m \leftarrow Rank_{desc}(P_m) \\ \bar{P}_b \leftarrow Rank_{desc}(P_b) \end{cases} \tag{24}$$

The per-class rankings for each base classifier are aggregated and then ranked again:

$$\begin{cases} f_k \leftarrow \bar{P}_{k,m} + \bar{P}_{k,b} \\ F \leftarrow f_k \end{cases} , \forall k \in \{1, ..., K\} \tag{25}$$

$$\bar{F} \leftarrow Rank_{desc}(F) \tag{26}$$

Finally, from the set $\bar{F}$ comprising k ordered values of F, we select the top Z rankings and use them to assign weights according to Eq. (20). Suppose the RAPC scheme is taken as $S3$, we can determine the class prediction for an instance x from Eq. (5). If $S3$=RAPC is used (in conjunction with another scheme) within a pair in the set expressed by Eq. (4), then equation Eq. (9) will be employed for the class prediction of the instance.

4) The Ranked aggregate of Average accuracy and Class Differential (RACD) scheme: With RACD, the ranking is directly proportional to the sum of the initial rankings of the average performance accuracies and the initial rankings of the difference in performance between the classes. This method is designed to assign a larger weight to the base classifiers with good initial overall accuracy that also have a relatively smaller difference in performance between the classes. The algorithm is described as follows.

Suppose, we take the RACD method as scheme $S4$, define a set $\bar{H}$ for ordered values with cardinality K. Given A, the set of computed average accuracies for each base classifier (determined in the AAB scheme) compute the class differential for each corresponding classifier as follows:

$$g_k \leftarrow |P_{k,m} - P_{k,b}| \;\; , k \in \{1, ..., K\} \tag{27}$$

Define $G \leftarrow g_k$, $\forall k \in \{1, ..., K\}$ as the ordered set of g_k values to which a ranking function $Rank_{ascen}(.)$ is applied to rank g_k in ascending order of magnitude:

$$\bar{G} \leftarrow Rank_{ascen}(G) \tag{28}$$

Then, for each base classifier, aggregate the values and apply the ranking function $Rank_{desc}(.)$:

$$\begin{cases} h_k \leftarrow A_k + G_k \\ H \leftarrow h_k \end{cases} , A_k \in \bar{A}, G_k \in \bar{G}, \forall k \in \{1, ..., K\} \tag{29}$$

$$\bar{H} \leftarrow Rank_{desc}(H) \tag{30}$$

Thus, $\bar{H}$ is the set containing the ranked values of H in descending order of magnitude. The top Z rankings are then used according to Eq. 20 to assign the weights.

C. Model complexity

As mentioned earlier, the base classifiers initial accuracies are estimated using a stratified N-fold cross validation technique. This procedure will be performed only once during training (on the training-validation set) and the preliminary predictions for all x instances in X for every base classifier will be determined from the procedure. The configurations (weights) computed from each algorithm is applied together with these initial (base classifier) predictions to re-classify each instance accordingly. Since level-2 training prediction of instances requires only re-classification using $\dot{V}(x)$, $\forall x \in X$, the time complexity for utilizing R level 2 algorithms to predict the classes of X instances using Eq. (5) will be given by $O(RX)$. The pairwise class predictions also involve re-classification, thus the complexity involved for predicting the class of X instances using Eq. (9) will be given by $O(JX)$ where $J = \binom{R}{2}$. Likewise, for the unweighted majority vote the complexity will be $O(X)$ as re-classification is involved also. Since we utilize unweighted majority vote and pairwise combinations for final model building (Eq. (17)) the total training time complexity in level-2 is therefore given by $O(X) + O(JX) = O((J + 1)X)$ where $J = \binom{R}{2}$ for the R level 2 ranking-based algorithms.

IV. INVESTIGATION METHODOLOGY

A. Automated static analyzer for feature extraction

The features used in the experimental evaluation of the DroidFusion system are obtained using an automated static analysis tool developed with Python. The tool enables us to extract permissions and intents from the application manifest file after decompiling with AXMLprinter2 (a library for decompiling Android manifest files). In addition, API calls are extracted through reverse engineering the .dex files by means of Baksmali disassembler. The static analyzer also searches for dangerous Linux commands from the application files and checks for the presence of embedded .dex, .jar, .so, and .exe files within the application. Previous works [35] have shown that these set of static application attributes provide discriminative features for machine learning based Android malware detection, hence, we utilized them for DroidFusion experiments. Furthermore, while extracting API calls, third party libraries are excluded using the list of popular ad libraries obtained from [36]. Fig. 2 shows an overview of the feature

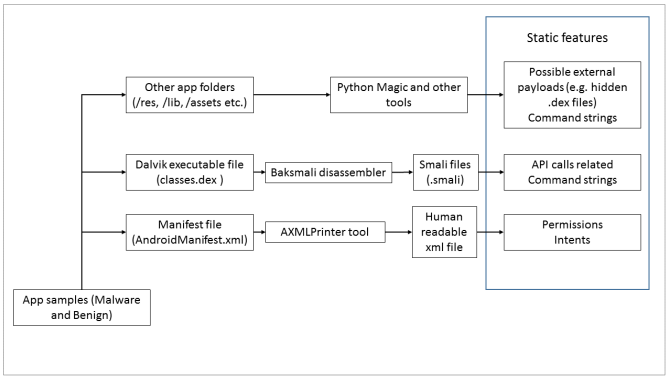

Fig. 2: Overview of the python based static analyzer for automated feature extraction.

extraction process using our the static app analyzer. The features are represented in binary form and labelled with class values in all the datasets.

B. Feature selection

Feature ranking and selection is usually applied for dimensionality reduction which in turn lowers model computational cost. The study in this paper utilized four datasets for evaluating DroidFusion. One of the datasets is derived from feature reduction of an initial set of 350 features down to 100 by applying the Information Gain (IG) feature ranking approach to rank the features and then selecting the top n features. IG evaluates the features by calculating the information gain achieved by each feature. Specifically, given a feature X, IG is expressed as:

$$IG = E(X) - E(X/Y) \tag{31}$$

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 feature X is given by:

$$E(X) = -\sum_{x \in X} p(x) log_2(p(x)) \tag{32}$$

Where $p(x)$ is the marginal probability density function for the random variable X. Similarly, the entropy of X relative to Y is given by [38]:

$$E(X/Y) = -\sum_{x \in X} p(x) \sum_{x \in X} p(x|y) log_2(p(x|y)) \tag{33}$$

Where $p(x|y)$ is the conditional probability of x given y. The higher the reduction of the entropy of feature X, the greater the significance of the feature.

C. Model Evaluation Metrics

The following performance metrics are considered in the evaluation of the models:

- True positive ratio (TPR): The ratio of correctly classified malicious apps to the total number of malicious apps. This is given by:

$$TPR = \frac{TP}{TP + FN} \tag{34}$$

Where TP (true positives) is the number of correct predictions of malware classification and FN (False negatives) is the number of misclassified malware instances in the set. TPR is also synonymous with $recall$ and $sensitivity$.

- False positive ratio (FPR): The ratio of incorrectly classified benign instances to the total number of benign instances, given by:

$$FPR = \frac{FP}{TN + FP} \tag{35}$$

Where FP (false positives) is the number of incorrect predictions of benign classifications while TN (true negatives) is the number of correct predictions of benign instances.

- Precision: also known as $positive\ predictive\ rate$ is calculated as follows:

$$Precision = \frac{TP}{TP + FP} \tag{36}$$

- F-measure: This metric combines precision and recall as follows:

$$FM = \frac{2 \cdot precision \cdot recall}{precision + recall} \tag{37}$$

In [20], is has been shown that (especially for unbalanced datasets) F-measure is a better metric than the Area Under Cure (AUC) for the Receiver Operating Cost (ROC) which uses values of TPR and FPR to plot a graph for different thresholds. Thus, in our experiments we utilize F-measure as the main indicator of predictive power. Note that precision and recall can be calculated for both malware and benign classes. Hence, if Fm and Fb are the F-measures for malware and benign classes respectively while N_m and N_b are the number of instances in each class, the combined metric known as weighted F-measure is the sum of F-measures weighted by the number of instances in each class, given by:

$$WFM = \frac{Fm \cdot N_m + Fb \cdot N_b}{N_m + N_b} \tag{38}$$

- Time taken to test the model. This is the time in seconds to test a constructed model from the testing set. All models were evaluated on a Windows 7 Enterprise 64 bit PC with 32GB of RAM and Intel Xeon CPU 3.10 GHz speed.

D. Datasets description

The experiments performed to evaluate DroidFusion was done using four datasets from three collections of Android app samples. Table II shows the details of each of the datasets. The first one (Malgenome-215) consists of feature vectors from 3,799 app samples where 2,539 were benign and 1,260 were malware samples from the Android malware genome project [3], a reference malware samples collection widely used by the malware research community. This dataset contains 215 features. The second dataset (Drebin-215) also consists of vectors of 215 features from 15,036 app samples; of these,

TABLE II: Datasets used for the DroidFusion evaluation experiments.

Datasets	#samples	#malware	#benign	#features
Malgenome-215	3799	1260	2539	215
Drebin-215	15036	5560	9476	215
McAfee-350	36183	13805	22378	350
McAfee-100	36183	13805	22378	100

9476 were benign samples while the remaining 5,560 were malware samples from the Drebin project [4]. The Drebin samples are also publicly available and widely used in the research community. Both Drebin-215 and Malgenome-215 datasets are made available as supplementary material.

The final two datasets come from the same source of samples. These are McAfee-350 and McAfee-100 in the table. They both have 36,183 instances of feature vectors derived from 13,805 malware samples and 22,378 benign samples made available to us by Intel Security (Formerly McAfee). Dataset #3 has 350 features, while Dataset #4 has the top 100 features with the largest information gain from the original 350 features in Dataset #3. In the experiments presented, Dataset #1, #2 and #3 are used to investigate DroidFusion with singular base classifiers, while Dataset #4 is used to study the fusion of ensemble base classifiers with DroidFusion. Note that all of the features were extracted using our static app analysis tool described in section IV-A.

V. Results and discussions

In this section, we present and discuss the results of four sets of experiments performed to evaluate DroidFusion performance. We utilized the open source Waikato Environment for Knowledge Analysis (WEKA) toolkit [37] to implement and evaluate DroidFusion. Feature ranking and reduction of dataset #3 into dataset #4 was also done with WEKA. In all the experiments we set K=5, i.e. five base classifiers are utilized. Also, we take N=10 and Z=3 for the cross validation and weight assignments respectively. In the first three sets of experiments, non-ensemble base classifiers were used, which were: J48, REPTree, Voted Perceptron and Random Tree. The Random Tree learner was used to build two separate classifier models using different configurations i.e. Random Tree-100 and Random Tree-9. With Random Trees, the number of variables selected at each split during tree construction is a configuration parameter which by default (in WEKA) is given by: $log_2 f + 1$, where f is the number of features (# variables = 9 for f =350 with the McAfee-350 dataset). The same configuration is used in the Drebin-215 and Malgenome-215 experiments for consistency. Thus, selecting 100 and 9 for Random Tree-100 and Random Tree-9 respectively results in two different base classifier models. Random Tree, REPTree, J48 and Voted Perceptron were selected as example base classifiers (out of 12 base classifiers) because of their combined accuracy and training time performance as determined from preliminary investigation; a different set of learning algorithms can be used with DroidFusion since it designed to be general-purpose, and not specific to a particular type of machine learning algorithm.

TABLE III: malgenome 215 train-validation set results and Level-2 algorithm based rankings for the base classifiers (5 = highest rank, 1 = lowest).

Classifier	TPR	TNR	AAB	CDB	RAPC	RACD
J48	0.975	0.983	4	4	5	5
REPTree	0.961	0.974	1	2	1	1
Random Tree-100	0.972	0.982	3	3	3	2
Random Tree-9	0.966	0.973	2	5	1	4
Voted Perceptron	0.971	0.991	5	1	4	2

TABLE IV: malgenome 215 train-validation set Level-2 combination schemes intermediate results.

Combination	PrecM	RecalM	PrecB	RecalB	W-FM
AAB+CDB	0.980	0.985	0.993	0.990	0.9883
AAB+RAPC	0.984	0.984	0.992	0.992	0.9893
AAB+RACD	0.982	0.984	0.992	0.991	0.9887
CDB+RAPC	0.982	0.984	0.992	0.991	0.9887
CDB+RACD	0.976	0.983	0.992	0.988	0.9864
RAPC+RACD	0.982	0.984	0.992	0.991	0.9887

A. Performance of DroidFusion with the Malgenome-215 dataset.

In order to evaluate DroidFusion on the Malgenome- 215 dataset, we split the dataset into two parts, one for testing and another for training-validation. The ratio was training-validation: 80%, testing: 20%. The stratified 10- fold cross validation approach was used to construct the DroidFusion model using the training-validation set. Table III shows the per-class accuracies of each of the 5 base classifiers resulting from 10-fold cross-validation on the training-validation set. The subsequent rankings determined from AAB, CDB, RAPC and RACD are also presented. Each of the algorithms induced a different set of rankings from the base classifiers accuracies. After applying Eq.(9) to the instances in the training-validation set and computing the accuracies with Eqs. (10)-(12), we obtained the performances of the pairwise combinations of the level-2 algorithms as shown in Table IV.

The results in Table IV clearly depict the overall performance improvement achieved by the level-2 combination schemes over the individual base classifiers. From Table III, J48 has the best malware recall of 0.975 but its recall for benign class is 0.983. On the other hand, Voted Perceptron had the best recall of 0.991 for the benign class, but its recall for the malware class is 0.971 (on the training-validation set). On the training-validation set, the best combination is AAB+RAPC (i.e. $S1S3$ pair) having 0.984 recall for malware and 0.992 recall for benign class, and a weighted F-measure of 0.9893. J48 and Voted Perceptron had weighted F-measures of 0.9804 and 0.9843 respectively. These were below all of the weighted F-measures achieved by the combination schemes shown in Table IV. Hence, these intermediate training-validation set results already show the capability of the DroidFusion approach to produce stronger models from the weaker base classifiers.

TABLE V: malgenome 215 Comparison of DroidFusion with base classifiers and traditional combination schemes on test set.

Classifier	PrecM	RecM	PrecB	RecB	W-FM	T(s)
J48	0.948	0.948	0.974	0.974	0.9654	0.02
REPTree	0.960	0.956	0.978	0.980	0.9720	0.01
Random Tree-100	0.967	0.956	0.978	0.984	0.9747	0.03
Random Tree-9	0.955	0.944	0.972	0.978	0.9667	0.02
Voted Perceptron	0.971	0.956	0.978	0.986	0.9760	0.05
Maj. voting	0.988	0.960	0.980	0.994	0.9827	0.05
Average of Probabilities	0.988	0.960	0.980	0.994	0.9827	0.06
Maximum Probability	0.906	0.988	0.994	0.949	0.9623	0.04
MultiScheme	0.983	0.956	0.978	0.992	0.9800	0.07
DroidFusion	**0.984**	**0.968**	**0.984**	**0.992**	**0.9840**	**0.07**

After the model has been built with the help of the training-validation set, the full DroidFusion model (featuring AAB+RAPC in level-2) was evaluated on the test set. For comparison, the base classifier models were re-trained on the complete training-validation set and then tested on the same test set. The results are shown in Table V. Figure 3, is a graph of the respective weighted F- measures. The results of DroidFusion are also compared to those of three classifier combination methods: Majority Vote, Maximum Probability and Average of Probabilities [13], and a meta learning method known as MultiScheme. The MultiScheme approach evaluates a given number of base classifiers in order to select the best model. In WEKA, it can be configured to use cross-validation or to build its model on the entire training set. In our experiments we selected 10-fold cross validation configuration for the MultiScheme learner to enable a comparative equivalent to DroidFusion. Time $T(s)$ depicts the testing time on the entire instances in the test set for each of the methods in Table V.

On the test set, Random Tree-100 recorded the best weighted F-measure out of the 5 base classifiers. Table V shows that higher precision, recall (for both classes) and a larger weighted F-measure was obtained with DroidFusion compared to all of the base classifiers. DroidFusion also performed better than MultiScheme and all the three combination schemes. These results with Malgenome-215 dataset demonstrate the effectiveness of the DroidFusion approach.

B. Performance of DroidFusion with the Drebin-215 dataset.

In this section, we present the evlaution of DroidFusion on the Drebin-215 dataset.Table VI shows the predictive accuracies on the 5 non- ensemble base classifiers on the training-validation set during DroidFusion model training. The split ratios for the training-validation and testing sets was 90%:10% and the 10-fold cross-validation procedure was utilized during training. The rankings induced by AAB, CDB, RAPC and RACD algorithms are also shown. Again, applying Eq. (9) to the instances in the training-validation set and computing accuracies with Eqs. (10)-(12) the performances of

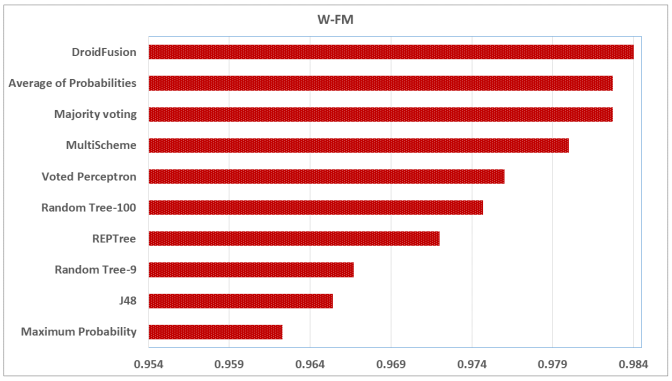

Fig. 3: Weighted F-measure results from the Malgenome-215 dataset experiments.

TABLE VI: DREBIN 215 train-validation set results and Level-2 algorithm based rankings for the base classifiers (5 = highest rank, 1 = lowest).

Classifier	TPR	TNR	AAB	CDB	RAPC	RACD
J48	0.959	0.983	4	3	5	4
REPTree	0.950	0.979	1	1	1	1
Random Tree-100	0.968	0.981	5	5	4	5
Random Tree-9	0.958	0.977	2	4	2	3
Voted Perceptron	0.956	0.982	3	2	3	2

the pairwise combinations of the level-2 algorithms are shown in Table VII.

From Table VI, Random Tree-100 had the best recall rate for the malware class (i.e. 0.968) while J48 had the best recall rate for the benign class (0.983). On the training-validation set, the weighted F-measure for Random Tree-100 was 0.9762, while that of J48 was 0.9741. Looking at Table VII, all of the combination schemes had better Weighted F-measures (than the base classifiers) indicating accuracy performance enhancement potential at this stage. The best combination is the RAPC+RACD ($S3S4$ pair) scheme, whose configuration is selected to build the final DroidFusion model.

After the full DroidFusion model was built, it was then evaluated on the test set. The base classifiers were re-trained on the entire training-validation set and tested on the test set for comparison. The results are presented in Table VIII, where Random Tree-100 can be seen to have the best Weighted F-measure (0.9824) out of the 5 base classifiers. The Droid-

TABLE VII: DREBIN 215 train-validation set Level-2 combination schemes intermediate results.

Combination	PrecM	RecalM	PrecB	RecalB	W-FM
AAB+CDB	0.966	0.972	0.984	0.980	0.9771
AAB+RAPC	0.984	0.972	0.984	0.991	0.9840
AAB+RACD	0.966	0.972	0.984	0.980	0.9771
CDB+RAPC	0.981	0.972	0.984	0.989	0.9827
CDB+RACD	0.966	0.972	0.984	0.980	0.9771
RAPC+RACD	0.981	0.976	0.986	0.989	0.9842

TABLE VIII: DREBIN 215 Comparison of DroidFusion with base classifiers and traditional combination schemes on test set.

Classifier	PrecM	RecM	PrecB	RecB	W-FM	T(s)
J48	0.972	0.964	0.979	0.984	0.9766	0.03
REPTree	0.976	0.951	0.972	0.986	0.9730	0.04
Random Tree-100	0.975	0.978	0.987	0.985	0.9824	0.04
Random Tree-9	0.947	0.971	0.983	0.968	0.9692	0.04
Voted Perceptron	0.969	0.950	0.971	0.982	0.9701	0.37
Maj. voting	0.983	0.973	0.984	0.990	0.9837	0.32
Average of Probabilities	0.983	0.973	0.984	0.990	0.9837	0.31
Maximum Probability	0.908	0.996	0.998	0.941	0.9617	0.33
MultiScheme	0.984	0.969	0.982	0.984	0.9784	0.05
DroidFusion	**0.981**	**0.984**	**0.991**	**0.989**	**0.9872**	**0.38**

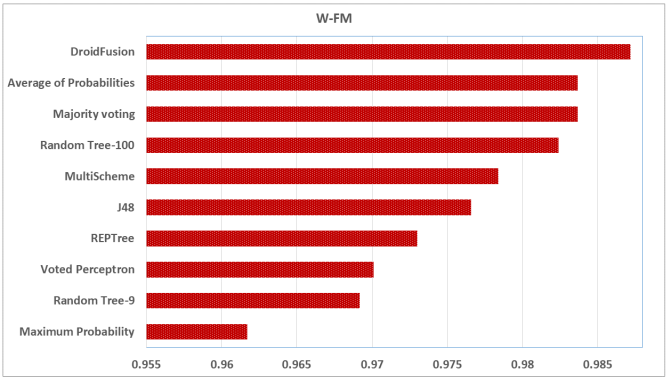

Fig. 4: Weighted F-measure results from the Drebin-215 dataset experiments.

Fusion model recorded the best precision and recall (for both classes) compared to the base classifiers resulting in a weighted F-measure of 0.9870. Figure 4 illustrates the graph of F-measures for the test set results. DroidFusion can be seen to also perform better than to Majority Vote, Maximum Probability, Average of Probabilities and MultiScheme. These results clearly demonstrate the effectiveness of the DroidFusion approach.

C. *Performance of DroidFusion with the McAfee-350 dataset.*

In this section, the results of experiments on the McAfee-350 dataset are presented. The same split ratios for training-validation/testing and the procedures applied in the previous experiment was adopted. The rankings from AAB, CDB, RAPC and RACD are shown in Table IX alongside the per-class accuracy performances on the validation set that induced the rankings. Just like in the previous experiments, we apply Eq.(9) to the instances in the training-validation set and compute the accuracies with Eqs. (10)-(12). The resulting performances of the pairwise combinations of the level-2 algorithms are shown in Table X.

From Table IX (training-validation set results for the base classifiers), J48 had the best benign class recall of 0.973

TABLE IX: McAfee train-validation set results and Level-2 algorithm based rankings for the base classifiers (5 = highest rank, 1 = lowest).

Classifier	TPR	TNR	AAB	CDB	RAPC	RACD
J48	0.941	0.973	4	3	4	3
REPTree	0.928	0.966	2	2	2	2
Random Tree-100	0.948	0.968	5	5	4	5
Random Tree-9	0.935	0.962	3	4	2	3
Voted Perceptron	0.917	0.959	1	1	1	1

TABLE X: McAfee 350 train-validation set Level-2 combination schemes intermediate results.

Combination	PrecM	RecalM	PrecB	RecalB	W-FM
AAB+CDB	0.945	0.955	0.972	0.966	0.9618
AAB+RAPC	0.970	0.956	0.973	0.982	0.9720
AAB+RACD	0.969	0.956	0.973	0.981	0.9714
CDB+RACD	0.969	0.955	0.972	0.981	0.9710
CDB+RACD	0.966	0.972	0.984	0.980	0.9771
RAPC+RACD	0.970	0.957	0.974	0.982	0.9724

amongst the 5 base classifiers. Random Tree-100 had the best malware class recall of 0.948 out of the 5 base classifiers. J48 had the highest Weighted F-measure of 0.9684. This is less than the Weighted F-measure of all combination schemes (shown in Table X) except the AAB+CDB scheme which had a Weighed F-measure of 0.9618. These intermediate results of the DroidFusion approach demonstrate the potential performance improvement obtainable in the final model.

Table XI shows the results of the base classifiers and the final DroidFusion model on the test set. The table, and the graphs in Figure 5 clearly show that DroidFusion increases performance accuracy over the single-algorithm base classifiers. DroidFusion results are equal to that of Majority vote and Average of Probabilities but perform better than the Maximum probability and MultiScheme methods. This is because, Eq. (17) selected mv as the strongest classifier over any of the pairs based on the computations on the initial N-fold cross validation predictions of the base classifiers. The mv scheme in this case achieved a W-FM of 0.9735 compared to 0.9724 obtained by CDB+RAPC ($S2S3$ pair) and RAPC+RACD ($S3S4$ pair). Therefore, DroidFusion was configured to use Eqs. (13)-(16) on the test set. However, if either of the strongest pairs had been used, it would result in a Weighted F-measure performance of 0.9777 on the test set; which still surpasses the Weighted F-measures from Maximum probability (0.9423) and those of the five original base classifiers. These results once again confirm the effectiveness of the proposed DroidFusion approach. In the next section we presents results obtained from experiments investigating ensemble learners as base classifiers.

TABLE XI: McAfee 350 Comparison of DroidFusion with base classifiers and traditional combination schemes on test set.

Classifier	PrecM	RecM	PrecB	RecB	W-FM	T(s)
J48	0.967	0.950	0.969	0.980	0.9685	0.11
REPTree	0.942	0.943	0.965	0.964	0.9560	0.11
Random Tree-100	0.954	0.951	0.970	0.972	0.9640	0.11
Random Tree-9	0.952	0.936	0.961	0.971	0.9576	0.12
Voted Perceptron	0.928	0.917	0.949	0.956	0.9411	6.76
Maj. voting	0.980	0.964	0.978	0.988	0.9788	6.76
Average of Probabilities	0.980	0.964	0.978	0.988	0.9788	7.01
Maximum Probability	0.874	0.990	0.993	0.912	0.9423	6.54
MultiScheme	0.967	0.950	0.969	0.980	0.9685	0.12
DroidFusion	**0.980**	**0.964**	**0.978**	**0.988**	**0.9788**	**7.02**

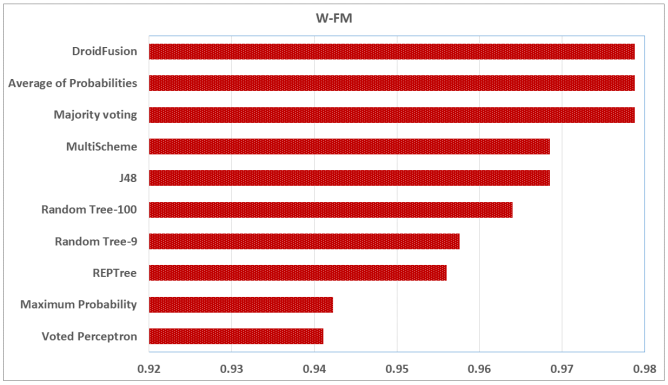

Fig. 5: Weighted F-measure results from the McAfee-350 dataset experiments.

D. Performance of DroidFusion with the McAfee-100 dataset using ensemble learners as base classifiers.

In this section we present results of experiments performed to investigate the feasibility of utilizing DroidFusion to enhance accuracy performance by combining ensemble classifiers rather than traditional singular classifiers. Ensemble learners have been shown to perform well in classification problems [14], [33]. Our goal is to investigate whether by using DroidFusion for fusion of ensemble classifiers, further accuracy improvements can be achieved. For our ensemble learning based experiments, we reduced the number of features from 350 down to 100 using the information gain feature ranking technique (Eq. 31-33). The ensemble learners considered as example base classifiers include: Random Forest [39], AdaBoost [40] (with Random Tree base classifier), Random Committee (with Random Tree base classifier), Random Subspace [41] (with Random Tree base classifier), and Random Subspace with REPTree base classifier. Note that the two Random Subspace learners with different base classifiers yield completely different models. In terms of number of iterations for the ensemble learners the configurations used were: AdaBoost (25 iterations), Random Forest, Random Committee, and Random Subspace (10 iterations each). Our choice of Random Tree as base learner for the ensemble (base) classifiers comes from our preliminary experiments (ommitted due to space constraint) which also confirms previous suggestion that it produces the strongest classifiers for most ensemble methods [42]. In the preliminary experiments, it was also found that by taking the top 100 features only a marginal drop in performance was observed for the ensemble base classifiers. Hence, this enabled us undertake the experiments with ensemble classifers using a significantly reduced dimension while using the same number of instances (i.e. 36,183).

TABLE XII: McAfee 100 train-validation set results and Level-2 algorithm based rankings for the (ensemble) base classifiers (5 = highest rank, 1 = lowest).

Classifier	TPR	TNR	AAB	CDB	RAPC	RACD
Random Forest	0.959	0.979	4	4	2	4
Random Sub. (REPTree)	0.923	0.984	1	1	2	1
AdaBoost: (Random Tree)	0.949	0.979	2	3	1	2
Random Sub. (Random Tree)	0.951	0.985	5	2	4	2
Random Comm. (Random Tree)	0.961	0.980	3	5	4	5

TABLE XIII: McAfee-100 train-validation set Level-2 combination schemes intermediate results.

Combination	PrecM	RecalM	PrecB	RecalB	W-FM
AAB+CDB	0.980	0.955	0.973	0.988	0.9753
AAB+RAPC	0.982	0.954	0.972	0.989	0.9756
AAB+RACD	0.980	0.955	0.973	0.988	0.9753
CDB+RAPC	0.980	0.955	0.973	0.988	0.9753
CDB+RACD	0.977	0.957	0.974	0.986	0.9749
RAPC+RACD	0.980	0.955	0.973	0.988	0.9753

Table XII shows the accuracy performance of the 5 ensemble models used as the DroidFusion base classifiers on the training-validation set instances (using the 10-fold cross-validation). The corresponding AAB, CDB, RAPC and RACD rankings are also depicted. Similar to the previous experiments, the level-2 combination schemes performance improves on that of the individual ensemble classifiers. This is also indicative of potential performance improvement obtainable when the final model is constructed. In this case, AAB+RAPC ($S1S3$ pair) is the recommended configuration as seen from Table XIII results.

In Table XIV, the test set results of the ensemble classifiers and those of DroidFusion are given. The results of MultiScheme, Majority vote, Average of probabilities and maximum probabilities are also shown. DroidFusion improves benign recall rates over all of the ensemble models in the base classifier level. The overall weighted F-measure of DroidFusion is the highest as shown in Table XIV and Figure 6 graphs. This shows that the DroidFusion approach can also be effectively applied for fusion of ensemble classifiers.

TABLE XIV: McAfee 100 Comparison of DroidFusion with (ensemble) base classifiers and traditional combination schemes on test set.

Classifier	PrecM	RecM	PrecB	RecB	W-FM	T(s)
Random Forest	0.960	0.965	0.978	0.975	0.9712	0.09
Random Sub. (REPTree)	0.971	0.931	0.958	0.983	0.9630	0.05
AdaBoost (Random Tree)	0.963	0.957	0.974	0.977	0.9694	0.11
Random Sub. (Random Tree)	0.974	0.957	0.974	0.984	0.9737	0.06
Random Comm. (Random Tree)	0.963	0.964	0.978	0.977	0.9720	0.08
Maj. voting	0.977	0.959	0.975	0.986	0.9757	0.24
Average of Probabilities	0.978	0.958	0.974	0.987	0.9759	0.19
Maximum Probability	0.972	0.958	0.974	0.983	0.9734	0.20
MultiScheme	0.960	0.968	0.980	0.975	0.9724	0.08
DroidFusion	**0.983**	**0.958**	**0.974**	**0.990**	**0.9777**	**0.22**

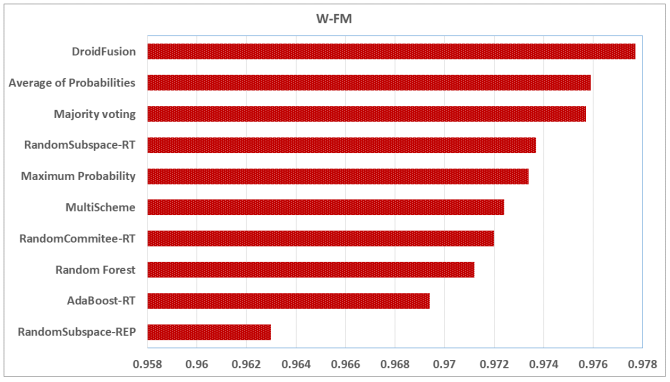

Fig. 6: Weighted F-measure results from the McAfee-100 dataset experiments with ensemble base classifiers.

E. Performance of DroidFusion vs. Stacked generalization.

Stacked Generalization [43], has a similar (multilevel) architecture to DroidFusion. It is also a well-known framework for classifier fusion which has been extensively studied and applied to many machine learning problems. For this reason, we compared our proposed approach to the Stacked Generalization method. One noticeable difference between our approach and Stacked Generalization is that instead of training with a meta- learner in level-2, we utilized a computational approach where ranking algorithms are used to combine the outcomes of the lower level classifiers. We used the StackingC implementation in WEKA which uses a Linear Regression meta classifier in level-2. Note that this is considered to be the most effective Stacked Generalization configuration [44] (given that any learning algorithm can be chosen as the meta classifier). The StackingC learner is also configured to use 10-fold cross validation when combining the base learners.

Applying the Stacked generalization algorithm to the same base classifiers and with the same four datasets the results are given in Figure 7 and Table XV. From Figure 7, the Weighted F-measures comparative results for the four datasets showed that StackingC achieved a better performance only in

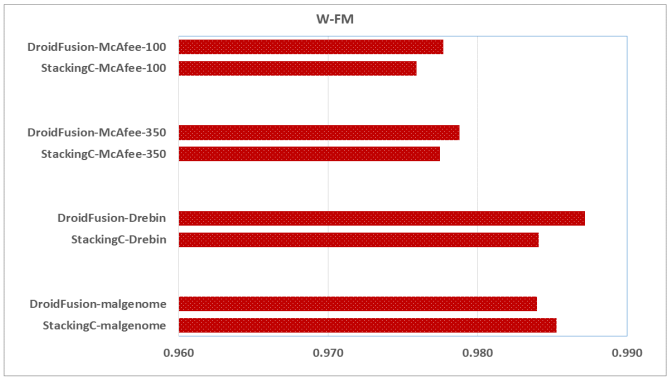

Fig. 7: DroidFusion vs. Stacking results (Weighted F-measure)

TABLE XV: DroidFusion vs. Stacked Generalization for the four datasets.

Method/dataset	PrecM	RecalM	PrecB	RecalB	W-FM
DroidFusion-malgenome	0.984	0.968	0.984	0.992	0.9840
StackingC-malgenome	0.992	0.964	0.982	0.996	**0.9853**
DroidFusion-Drebin	0.981	0.984	0.991	0.989	**0.9872**
StackingC-Drebin	0.988	0.969	0.982	0.993	0.9841
DroidFusion-McAfee-350	0.980	0.964	0.978	0.988	**0.9788**
StackingC-McAfee-350	0.974	0.967	0.980	0.984	0.9775
DroidFusion-McAfee-100	0.983	0.958	0.974	0.990	**0.9777**
StackingC-McAfee-100	0.978	0.958	0.974	0.987	0.9759

the case of the malgenome-215 dataset. On all the other three datasets, DroidFusion performed better. A notable advantage of DroidFusion over Stacking is that it provides a wider range of criteria for weighting and fusion of base classifiers through the use of four separate algorithms; by contrast, Stacking (with liner regression meta classifier) effectively combines classifiers based on only one criterion (i.e. weighting the base classifiers according to their relative strengths (overall performance accuracies) [44]).

F. Analysis of time performance

As mentioned earlier, the app processing to extract features was done using our bespoke Python based tool described in section IV-A. Table XVI presents an overview of app processing time estimates. This is dependent on the size of the app which can range between a few kilobytes to a several megabytes. Hence, the average unzipping and disassembly time was 0.739 seconds while the average time to analyse the manifest and extract permissions, intents etc. was 0.0048 seconds. The rest of the processing involves mining the disassembled files and scanning for other attributes. This took on average 6.4 seconds. The total average processing time for the apps was therefore approximately 7.145 seconds. During the experiments the feature vectors were fed into trained models for testing. The DroidFusion model testing times were 0.07

TABLE XVI: Analysis of app processing time

Task	Lowest (s)	Highest (s)	Average (s)
Unzipping and dissassemby	0.392	1.18	0.739
Manifest analysis	0.0013	0.0088	0.0048
Code analysis	3.428	15.47	6.4
Total			**7.145**

seconds (for 759 instances), 0.38 seconds (for 1503 instances), 7.02 seconds (for 3618 instances), and 0.22 seconds (for 3618 instances) in the four sets of experimental results presented earlier. These figures clearly illustrate the scalability of static-based features solution with only an average of just over 7 seconds required to process an app and classify it using a trained DroidFusion model. Thus, it is feasible in practice to deploy the system for scenarios requiring large scale vetting of apps.

Note that although our study is based on specific static features, classifiers trained from other types of features can also be combined using DroidFusion. Basically, DroidFusion is agnostic to the feature engineering process.

G. Limitations of DroidFusion

Although the proposed general-purpose DroidFusion approach has been demonstrated empirically to enable improved accuracy performance by classifier fusion, there is scope for further improvement. The current DroidFusion design is aimed at binary classification. Future work could investigate extending the algorithms in the DroidFusion framework to handle multi-class problems.

VI. Conclusion

In this paper, we proposed a novel general purpose multi-level classifier fusion approach (DroidFusion) for Android malware detection. The DroidFusion framework is based on four proposed ranking-based algorithms that enable higher-level fusion using a computational approach rather than the traditional meta classifier training that is used for example in Stacked Generalization. We empirically evaluated DroidFusion using four separate datasets. The results presented demonstrates its effectiveness for improving performance using both non-ensemble and ensemble base classifiers. Furthermore, we showed that our proposed approach can outperform Stacked Generalization whilst utilizing only computational processes for model building rather than training a meta classifier at the higher level.

Acknowledgment

This work is supported by the UK Engineering and Physical Sciences Research Council EPSRC grant EP/N508664/1 Centre for Secure Information Security (CSIT-2).

References

[1] Smartphone OS market share worldwide 2009-2015 Statistic, Statista, Hamburg, Germany, 2017 [Online] https://www.statista.com/statistics/263453/global-market- share-held-by-smartphone-operating-systems

[2] McAfee Labs. McAfee Labs Threat Predictions Report. March 2016.

[3] Y. Zhou and X. Jiang, "Dissecting android malware: Characterization and evolution" In proc. 2012 IEEE Symposium on Security and Privacy (SP), San Fransisco, CA, USA, 20-23 May, 2012 , pp. 95-109.

[4] D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, and K. Rieck, "Drebin: Efficient and Explainable Detection of Android Malware in Your Pocket" In proc. 20th Annual Network & Distributed System Security Symposium (NDSS), San Diego, CA, USA, 23-26 Feb. 2014.

[5] A. Apvrille, and R. Nigam. Obfuscation in Android Malware, and how to fight back. Virus Bulletin, July 2014. Available from: https://www.virusbulletin.com/virusbulletin/2014/07/obfuscation-android-malware-and-how-fight-back [Accessed Sept. 2017]

[6] Y. Jing, Z. Zhao, G.-J. Ahn, and H. Hu, "Morpheus: Automatically Generating Heuristics to Detect Android Emulators" In proc. 30th Annual Computer Security Applications Conference (ACSAC 2014),New Orleans, Louisiana, USA, Dec. 8-12, 2014, pp. 216-225.

[7] T. Vidas and N. Christin, "Evading Android runtime analysis via sandbox detection" In proc. 9th ACM Symposium on Information, Computer and Communications Security, Kyoto, Japan, June 04-06, 2014, pp. 447-458.

[8] T. Petsas, G. Voyatzis, E. Athanasopoulos, M. Polychronakis, and S. Ioannidis, "Rage against the virtual machine: hindering dynamic analysis of android malware" In proc.7th European Workshop on System Security (EuroSec '14), Amsterdam, Netherlands, April 13, 2014.

[9] F. Matenaar and P. Schulz. Detecting android sandboxes. http://dexlabs.org/blog/btdetect, August 2012. [Accessed: Sept. 2017].

[10] S. R. Choudhary, A. Gorla, A. Orso, "Automated test input generation for Android: are we there yet?" In proc. 30th IEEE/ACM international conference on Automated Software Engineering (ASE 2015), Nov. 9-13, 2015, pp. 429-440.

[11] W. Dong-Jie, M. Ching-Hao, W. Te-En, L. Hahn-Ming, and W. Kuo-Ping, "DroidMat: Android malware detection through manifest and API calls tracing," In proc. Seventh Asia Joint Conference on Information Security(Asia JCIS), 2012, pp. 62-69.

[12] S. Y. Yerima, S. Sezer, and I. Muttik. "Android malware detection: An eigenspace analysis approach" In proc. Science and Information Conference (SAI 2015), London, UK, 28-30 July 2015, pp.1236-1242.

[13] S. Y. Yerima, S. Sezer, I. Muttik "Android malware detection using parallel machine learning classfiers" In proc. 8th Int. Conf. on Next Generation Mobile Apps, Services and Technolgies (NGMAST 2014), Oxford, UK, Sept. 10-12, 2014, pp. 37-42

[14] S. Y. Yerima, S. Sezer, and I. Muttik. High accuracy Android malware detection using ensemble learning. IET Information Security, Vol 9, issue 6, 2015, pp. 313-320.

[15] M. Varsha, P. Vinod, & K. Dhanya. Identification of malicious Android app using manifest and opcode features. Journal of Computer Virology and Hacking Techniques, 2016, pp. 1-14.

[16] A. Sharma and S. Dash, "Mining API calls and permissions for Android malware detection" in Cryptology and Network Security. Springer International Publishing, 2014, pp. 191205.

[17] P.P. K., Chan and W-K. Song, "Static detection of Andoid malware by using permissions and API calls" In proc. 2014 international Conference on Machine Learning and Cybernetics, Lanzhou, July 13-16, 2014.

[18] W. Wang, X. Wang, D. Feng, J. Liu, Z. Han and X. Zhang. Exploring Permission-Induced Risk in Android applications for Malicious Applcication Detection. IEEE Transactions on Information Forensics and Security, Vol. 9, No. 11, Nov. 2014, pp. 1869-1882.

[19] M. Fan, J. Liu, W. Wang, H. Li, Z. Tian and T. Liu. DAPASA: Detecting Android Piggybacked Apps through Sensitive Subgraph Analysis. IEEE Transactions on Information Forensics and Security, Vol. 12, Issue 8, March 2016, pp. 1772-1785.

[20] L. Cen, C. S. Gates, L. Si, and N. Li. A Probabilistic Discriminative Model for Android Malware Detection with Decompiled Source code. IEEE Transactions on Secure and Dependable Computing, Vol. 12, No. 4, July/August 2015.

[21] Westyarian, Y. Rosmansyah, B. Dabarsyan, "Malware detection on Android Smartphones using API class and Machine learning" 2015 International Conference on Electrical Enginnering and Informatics (ICEEI 2015), 10-11 Aug. 2015.

[22] F. Idrees, and M. Rajarajan. "Investigating the Android intents and permissions for malware detection". In proc. 10th IEEE Int. Conf. on Wireless and Mobile Computing, Networking and Communications (WiMob), Oct. 2014, pp. 354-358).

[23] B. Kang, S. Y. Yerima, S. Sezer and K. McLaughlin. N-gram opcode analysis for Android malware detection. International Journal of Cyber Situational Awareness, Vol. 1, No. 1, Nov. 2016.

[24] M. Zhao, F. Ge, T. Zhang, and Z. Yuan, "Antimaldroid: An efficient svm based malware detection framework for android" In C. Liu, J. Chang, and

A. Yang, editors, ICICA (1), volume 243 of Communications in Computer and Information Science, Springer, 2011. pp. 158166.
[25] W.-C. Wu and S.-H. Hung, "Droiddolphin: A dynamic Android malware detection framework using big data and machine learning" In proc. 2014 ACM conf. on Research in Adaptive and Convergent Systems, (RACS '14), NY, USA, pp. 247-252.
[26] V. M. Afonso, M. F. de Amorim, A. R. A. Gregio, G. B. Junquera, and P. L. de Geus. Identifying Android malware using dynamically obtained features. Journal of Computer Virology and Hacking Techniques, 2014.
[27] M. K Alzaylaee, S. Y. Yerima, S. Sezer "EMULATOR vs REAL PHONE: Android Malware Detection Using Machine Learning" 3rd ACM Int. Workshop on Security and Privacy Analytics (IWSPA '17), Co-located with ACM CODASPY 2017, Scotts., AZ, USA, March 2017.
[28] Lindorfer, M., Neugschwandtner, M., & Platzer, C. "MARVIN: Efficient and comprehensive mobile app classification through static and dynamic analysis" In proc. IEEE 39th Annual Computer Software and Applications Conference (COMPSAC), pp. 4223433.
[29] D. Gaikwad and R. Thool "DAREnsemble: Decision Tree and Rule Learner Based Ensemble for Network Intrusion Detection System" In Proc. 1st Int. Conf. on Information and Communication Technology for Intelligent Systems, Springer, 2016, pp. 185-193.
[30] A. Balon-Perlin and B. Gamback. Ensemble of Decision Trees for Network Intrusion Detection. International Journal on Advances in Security, Vol. 6. No. 1 and 2, 2013.
[31] M. Panda and M. R. Patra "Ensembling rule based classifiers for detecting network intrusions" Int. Conf.on Advances in Recent Technologies in Communication and Computing, 2009, IEEE, DOI 10.1109/ARTCom.2009.121.
[32] A. Zainal, M. A. Maarof, S. M. Shamsuddin and A. Abraham Ensemble of one-class classifiers for network intrusion detection system In proc. Fourth international conference on information assurance and security, 2008, IEEE, DOI 10.1109/IAS.2008.35
[33] L. D. Coronado-De-Alba, A. Rodriguez-Mota, P. J. Escamilla- Ambrosio Feature Selection and ensemble of classifiers for Android malware detection In proc. 8th IEEE Latin-American Conference on Communications (LATINCOM 2016), 15-17 Nov. 2016.
[34] M. K Alzaylaee, S. Y. Yerima, S. Sezer "Improving Dynamic Analysis of Android Apps Using Hybrid Input Test Generation" In proc. Int. Conf. on CyberSecurity and Protection of Digital Services (Cyber Security 2017), London, UK, June 19-20, 2017.
[35] Y. Aafer, W. Du, and H. Yin, "DroidAPIMiner: Mining API-level features for robust malware detection in Android" In proc. 9th Int.Conference on Security and Privacy in Communication Networks (SecureComm 2013). Sydney, Australia, Sep. 25-27, 2013.
[36] T. Book, A. Pridgen, and D. S. Wallach, "Longitudinal Analysis of Android Ad Library Permissions" In proc. Mobile Security Technologies conference (MoST 13), San Fransisco, CA, May 2013.
[37] M. Hall, E. Frank, G. Holmes, B. Pfahriger, P. Reutermann and I. H. Witten. The WEKA data mining software: an update. ACM SIGKDD Explorations, Vol.11, No.1. June 2009.pp 10-18.
[38] T. M. Cover, J. A. Thomas, Elements of Information Theory, 2nd Edition, John Wiley & Sons, inc., Hoboken, New Jersey, 2006, pp. 41.
[39] L. Breiman. Random forests. Machine Learning, 45, 2001, pp 5-32.
[40] Y. Freund and R. E. Schapire, "Experiments with a new boosting algorithm" In proc. 13th Int. Conf. on Machine Learning, San Francisco, 1996, pp. 148-156.
[41] T. K. Ho. The Random Subspace Method for Constructing Decision Forests. IEEE Transactions on Pattern Analysis and Machine Intelligence. Vol 20 (8), 1998, pp. 832-844, 1998
[42] T. Ho, "Random Decision Forests", Proc. of the 3rd Int. Conf. on Document Analysis and Recognition, 1995, pp. 278-282.
[43] David H. Wolpert. Stacked generalization. Neural Networks. 1992, pp, 241-259.
[44] K. M. Ting and I. H. Witten. Issues in Stacked Generilization. Journal of Artificial Intelligence Research, 10, May 1999, pp. 271-289.
[45] T. Ban, T. Takahashi and S. Guo "Integration of Multi-modal Features for Android Malware Detection Using Linear SVM" In proc. 11th Asia Joint Conference on Information Security, 2016.
[46] Z. Ni, M. Yang, Z. Ling, J. N. Wu and J. Luo, "Real-Time Detection of Malicious Behavior in Android Apps," In proc. Int. Conf. on Advanced Cloud and Big Data (CBD), Chengdu, 2016, pp. 221-227.
[47] Z. Wang, J. Chai, S. Chen and W. Li, "DroidDeepLearner: Identifying Android malware using deep learning" IEEE 37th Sarnoff Symposium, Newark, NJ, 2016, pp. 160-165.
[48] S. Wu, P. Wang, X. Li, Y. Zhang. Effective detection of android malware based on the usage of data flow APIs and machine learning. Information and Software Technology, Vol.75, 2016, Pages 17-25, ISSN 0950-5849.
[49] M.-Y. Su, J.-Y. Chang, and K.-T. Fung "Machine Learning on Merging Static and Dynamic Features to identify malicious mobile apps" In proc. 9th Int. Conf. on Ubiquitous and Future Networks (ICUFN), 2017, Milan, Italy, 4-7 July 2017. pp. 863-867.
[50] N. Milosevic, A. Dehghantanha, K.-K. R. Choo "Machine Learning aided Android malware classification" Computers & Electrical Engineering, Volume 61, July 2017, pp 266-274.
[51] W. Wang, Y. Li, X. Wang, J. Liu, X. Zhang "Detecting Android malicious apps and categorizing benign apps with ensemble classifiers" Future Generation Computer Systems, 2017, ISSN 0167-739X.
[52] X. Wang, W. Wang, Y. He, J. Liu, Z. Han, X. Zhang "Characterizing Android apps behaviour for effective detection of malapps at large scale" Future Generation Computer Systems, Volume 75, Oct. 2017, pp. 30-45.
[53] A. Mahindru and P. Singh "Dynamic Permissions based Android malware detection using machine learning techniques" In proc. 10th Innovations in Software Engineering Conference, Jaipur, India, Feb. 5-7, 2017. pp 202-210.
[54] M. Yang, S. Wang, Z. Ling,, Y. Liu, Z. Ni. Detection of malicious behaviour in Android apps through API calls and permission uses analysis. Concurrency Computed: Pract Exper. 2017: e4172. https://doi.org/10.1002/cpe.4172
[55] F. Idrees, M. Rajarajan, M. Conti, T. M. Chen, Y. Rahulamathavan. PIndroid: A novel Android malware detection system using ensemble learning methods. Computers & Security, Vol 68, July 2017, pp. 36-46.

Suleiman Y. Yerima (M'04) received the B.Eng. degree (first Class) in electrical and computer engineering from the Federal University of Technology, Minna, Nigeria, the M.Sc, degree (with distinction) in personal, mobile and satellite communications from the University of Bradford, Bradford, U.K., and the Ph.D. degree in mobile computing and communications from the University of South Wales, Pontypridd, U.K. (formerly, the University of Glamorgan) in 2009.

He is currently a Senior Lecturer of Cyber Security in the Faculty of Technology, at De Montfort University, Leicester, United Kingdom. He was previously a Research Fellow at the Centre for Secure Information Technologies (CSIT), Queens University Belfast, UK, where he led the mobile security research theme from 2012 until 2017. He was a member of the Mobile Computing Communications and Networking (MoCoNet) Research group at Glamorgan from 2005 to 2009. From 2010 to 2012, he was with the UK- India Advanced Technology Centre of excellence in Next Generation Networks, Systems and Services (IU-ATC), University of Ulster, Coleraine, Northern Ireland .

Dr. Yerima is a member of the IAENG, and (ISC)2 professional societies. He is a Certified Information Systems Security Professional (CISSP) and a Certified Ethical Hacker (CEH). He was the recepient of the 2017 IET Information Security premium (best paper) award.

Sakir Sezer (M '00) received the Dipl. Ing. degree in electrical and electronic engineering from RWTH Aachen University, Germany, and the Ph.D. degree in 1999 from Queens University Belfast, U.K. Prof. Sezer is currently Secure Digital Systems Research Director and Head of Network Security Research in the School of Electronics Electrical Engineering and Computer Science at Queens University Belfast. His research is leading major (patented) advances in the field of high-performance content processing and is currently commercialized by Titan IC Systems. He has co- authored over 120 conference and journal papers in the area of high-performance network, content processing, and System on Chip. Prof. Sezer has been awarded a number of prestigious awards including InvestNI, Enterprise Ireland and Intertrade Ireland innovation and enterprise awards, and the InvestNI Enterprise Fellowship. He is also cofounder and CTO of Titan IC Systems and a member of the IEEE International System- on-Chip Conference executive committee.