DroidNMD: Network-based Malware Detection in Android Using an Ensemble of One-Class Classifiers

Fariba Ghaffari, Mahdi Abadi, Asghar Tajoddin, and Mahsa Lamiyan

***Abstract*—During the past few years, the number of malware designed for Android devices has increased dramatically. To confront with Android malware, some anomaly detection techniques have been proposed that are able to detect zero-day malware, but they often produce many false alarms that make them impractical for real-world use. In this paper, we address this problem by presenting DroidNMD, an ensemble-based anomaly detection technique that focuses on the network behavior of Android applications in order to detect Android malware. DroidNMD constructs an ensemble classifier consisting of multiple heterogeneous one-class classifiers and uses an ordered weighted averaging (OWA) operator to aggregate the outputs of the one-class classifiers. Our work is motivated by the observation that combining multiple one-class classifiers often produces higher overall classification accuracy than any individual one-class classifier. We demonstrate the effectiveness of DroidNMD using a real dataset of Android benign applications and malware samples. The results of our experiments show that DroidNMD can detect Android malware with a high detection rate and a relatively low false alarm rate.**

***Index Terms*—Android malware, anomaly detection, ensemble classifier, network behavior, one-class classifier, ordered weighted averaging.**

I. Introduction

Android is a mobile operating system that is based on the Linux kernel and designed primarily for touchscreen mobile devices such as smartphones and tablets. The development of mobile applications on Android has attracted a lot of attention in research and industry in recent years. Android applications come in the form of an Android package, which is an archive file that contains the compiled code and manifest file, as well as various resources and folders. More specifically, the Android compiler suite compiles the developer's Java files into class files, and then the class files are converted into DEX files. A DEX file contains the bytecode for the Dalvik virtual machine. The DEX files, XML files, and other resources that are required to run an Android application, are packaged into an Android package file with a .apk extension. Once the package is generated, it is signed with the developer's key and uploaded onto the Google Play store, formerly known as the Android Market.

According to a recent report [1], the number of available applications on Google Play, reached 2.8 million applications in March 2017, after surpassing 1 million applications in July 2013. In addition, Android devices are currently dominating the global smartphone market [2]. The growing popularity of Android applications and devices has attracted malware writers to exploit Android vulnerabilities and write more malware for it. Fig. 1 shows a growing trend in the number of new Android malware samples from 2012 to the first quarter of 2017 [3]. This growth is due to the fact that Android applications are often stored in third-party markets that do not analyze them against malicious code.

Existing Android malware detection techniques can be categorized into misuse detection and anomaly detection. Misuse detection techniques rely on identifying signatures of known malware, while anomaly detection techniques try to detect deviations from normal application behavior. The advantages of misuse detection techniques are that they produce low false alarms and that they can quickly detect malware. Their disadvantages are that they are not capable of detecting unknown or zero-day malware and that they have to maintain a large database of malware signatures. In contrast, anomaly detection techniques can potentially detect unknown malware, but they usually produce many false alarms.

Misuse and anomaly detection techniques generally rely either on static analysis or on dynamic analysis to extract particular features from malware samples. Static analysis [4] aims to analyze the source code of applications for suspicious patterns without actually running the applications in an Android emulator or device. A number of static analysis techniques [5] parse the manifest file for information such as requested permissions, services, broadcast receivers, and so on. On the other hand, dynamic analysis techniques [4] run applications in a controlled environment to reveal their runtime behavior. Static analysis is advantageous on resource-limited Android devices because it does not need to run applications. However, this technique has a major drawback of code obfuscation [6] and dynamic code loading [7]. In contrast, dynamic analysis is less vulnerable to various code obfuscation and dynamic code loading, but needs more resources and provides less code coverage [8]. Since some malicious behavior is only triggered by certain external events, dynamic analysis will not detect an application's malicious behavior if the events are not triggered.

Manuscript received July 14, 2017; accepted September 7, 2017.
The authors are with the Faculty of Electrical and Computer Engineering, Tarbiat Modares University, Tehran, Iran (e-mail: {fariba.ghaffari, abadi, a.tajoddin, m.lamiyan}@modares.ac.ir).

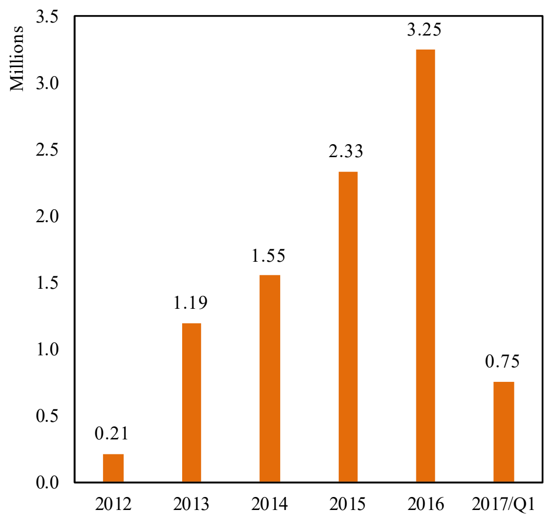

Fig. 1. Number of new Android malware samples (in Millions) from 2012 to the first quarter of 2017 [3].

In this paper, we propose DroidNMD, a novel network-based anomaly detection technique that uses an ensemble classifier consisting of multiple heterogeneous one-class classifiers to detect Android malware. DroidNMD focuses on the network behavior of Android applications. Specifically, the behavior of an application is represented by a set of features that are extracted from packets sent or received by the application. These features are divided into two groups: aggregation-based and entropy-based. Aggregation-based features measure the amount of network traffic consumed by the application, while entropy-based features measure the degree of complexity in the application's network behavior.

DroidNMD aggregates the outputs of one-class classifiers in the ensemble classifier using a novel ordered weighted averaging (OWA) operator, called NOWA, that increases the detection performance. It has been both theoretically and empirically demonstrated that ensemble classifiers can largely improve the classification accuracy of their constituent members [9]. Also, the OWA operators [10] are general aggregation operators that take multiple values as input and reorder them in ascending or descending order before aggregating them. Because of their simplicity and robustness, they have been widely used in ensemble classification [9], [11].

The rest of this paper is organized as follows: Section II reviews the related work. Section III is fully dedicated to the background. Section IV presents DroidNMD and Section V reports experimental results. Finally, Section VI draws some conclusions.

II. Related Work

In response to the growing number of Android malware, researchers have introduced various detection techniques that can be categorized into misuse detection and anomaly detection. In the following, we give a brief overview of these techniques.

Aafer *et al.* [12] present a misuse detection technique for Android malware that relies on API level information within the bytecode. They first extract API calls and their package level information as features from both benign applications and malware samples, and then reduce the feature set to contain only those API calls whose support in malware samples is significantly higher than in benign applications. Finally, they generate a set of feature vectors and use them to build different classification models. Arp *et al.* [13] propose another misuse detection technique for Android malware that performs static analysis to extract 8 different feature sets (such as hardware features, requested permissions, suspicious API calls, and so on) from benign applications and malware samples. The feature sets are then mapped to a joint vector space and used to train a linear SVM classifier. Feng *et al.* [14] present Apposcopy, a semantics-based approach for identifying a prevalent class of Android malware that steals private and peronal user information. Apposcopy incorporates a high-level language for specifying signatures that describe semantic characteristics of malware families and a static analysis for deciding if a given application matches a malware signature. Yang *et al.* [15] consider a security-sensitive behavior as an invocation of a security-sensitive method under a certain context and introduce AppContext that uses static analysis to differentiate between benign and malicious behaviors within Android applications based on the contexts that trigger security-sensitive behaviors. A method is security-sensitive if it is permission-protected or it is either a source method or a sink method of an information flow. Dash *et al.* [16] propose DroidScribe, a misuse detection technique that uses a multi-class SVM classifier to classify Android malware into families based on runtime behavior derived from system call traces. DroidScribe refines the SVM classification by selectively applying conformal prediction to obtain sets of matches whenever the SVM classifier does not achieve an acceptable confidence. Wang *et al.* [17] extract 34,630 static features from each Android application and employ an ensemble of multiple classifiers to detect Android malware and to categorize benign applications.

Another active area of research deals with anomaly detection in Android. Shabtai *et al.* [18] attempt to detect a specific type of Android malware with self-updating capabilities that accomplish malicious activities sometime after the installation (e.g., after dynamic loading of a precompiled code). For each application, semi-supervised machine learning algorithms are used to detect meaningful deviations from its expected network behavior. However, their experimental results show that this technique has relatively low detection rate on applications infected with Trojans, where the main functionality is preserved and some new functionality is added. Ghaffari *et al.* [19] propose DroidMalHunter, an entropy-based anomaly detection system that focuses on profiling applications' network behavior to detect Android malware. DroidMalHunter uses a one-class SVM classifier to identify applications that have anomalous network behavior, and then calculates an anomaly score for each application according to its current and previous suspicious values. An application is reported as malware if its anomaly score exceeds a predefined threshold. Saracino *et al.* [20] propose MADAM, a behavior-based and multi-level malware detection system for Android devices that concurrently monitors running applications by retrieving five groups of features at four different levels of abstraction. For some groups of features, MADAM applies an anomaly detection technique and for the others it implements a misuse detection technique that considers behavioral patterns obtained from known malware misbehaviors. However, their usability experiments show that the number of false positives per day noticeably increases with a heavy usage. In particular, a large number of false positives are generated by installation of new applications.

III. Background

In this section, we give a brief overview of some basic concepts and techniques used throughout this paper.

A. Modified Sample Entropy

The modified sample entropy [21], also known as mSampEn, is a conditional probability measure that quantifies the likelihood that two sequences of the same length m, matching each other within a tolerance of r, will continue to be matched when their length is increased to $m+1$. More formally, given a finite sequence $X = \langle x_1, x_2, \ldots, x_N \rangle$, let X_i^m, $i = 1, 2, \ldots, N-m+1$, be subsequences of X with length m that are generalized by removing a local baseline:

$$X_i^m = \langle x_i - \mu_i^m, x_{i+1} - \mu_i^m, \ldots, x_{i+m-1} - \mu_i^m \rangle\,, \qquad (1)$$

where μ_i^m is defined as

$$\mu_i^m = \frac{1}{m} \sum_{k=0}^{m-1} x_{i+k}\,. \qquad (2)$$

The distance between any two subsequences X_i^m and X_j^m ($i, j = 1, 2, \ldots, N-m$, $i \neq j$), denoted by d_{ij}^m, is defined as

$$d_{ij}^m = \max_{0 \le k \le m-1} \left| x_{i+k} - x_{j+k} \right|. \qquad (3)$$

Then, the probability of matching any two subsequences of length m is given by

$$B^m(r) = \frac{1}{N-m} \sum_{i=1}^{N-m} B_i^m(r)\,, \qquad (4)$$

where $B_i^m(r)$ is the probability that any subsequence X_j^m is matched with a subsequence X_i^m within r:

$$B_i^m(r) = \frac{1}{N-m-1} \sum_{j=1, j\neq i}^{N-m} \vartheta\left(r, d_{ij}^m\right), \qquad (5)$$

where ϑ is a sigmoid-based similarity function between any two subsequences X_i^m and X_j^m:

$$\vartheta\left(r, d_{ij}^m\right) = \frac{1}{1 + \exp\left(\left(d_{ij}^m - 0.5\right)/r\right)}\,. \qquad (6)$$

It should be mentioned that in (5) and (6), only the first $N-m$ subsequences of length m are considered to ensure that, for all $i = 1, 2, \ldots, N-m$, both X_i^m and X_i^{m+1} are defined.

Similarly, the probability of matching any two subsequences of length $m+1$ is given by

$$A^m(r) = \frac{1}{N-m} \sum_{i=1}^{N-m} A_i^m(r)\,, \qquad (7)$$

where $A_i^m(r)$ is defined as

$$A_i^m(r) = \frac{1}{N-m-1} \sum_{j=1, j\neq i}^{N-m} \vartheta\left(r, d_{ij}^{m+1}\right). \qquad (8)$$

Finally, the modified sample entropy (mSampEn) of X, denoted by mSampEn(X, m, r), is defined as

$$\text{mSampEn}(X, m, r) = -\ln\left(A^m(r)/B^m(r)\right). \qquad (9)$$

B. Ordered Weighted Averaging Operators

The ordered weighted averaging (OWA) operators [10] are general aggregation operators in which the order of the arguments has primary role in the aggregation process. Formally, an OWA operator of dimension L is a mapping $\mathcal{O}_{\mathbf{w}}: \mathbb{R}^L \to \mathbb{R}$ which has an associated weighting vector $\mathbf{w} = (w_1, w_2, \ldots, w_L)$ in which $\sum_{i=1}^{L} w_i = 1$ and $0 \le w_i \le 1$ for all $i = 1, 2, \ldots, L$, such that

$$\mathcal{O}_{\mathbf{w}}(a_1, a_2, \ldots, a_L) = \sum_{i=1}^{L} w_i b_i\,, \qquad (10)$$

where b_i is the ith smallest value among all the arguments a_1, $a_2, \ldots, a_L$. Note that here we assume that the elements of $\mathbf{w}$ are in ascending order.

The result of the aggregation performed by an OWA operator mainly depends upon its associated weights. To characterize the associated weights of an OWA operator, we can use the orness measure, which is defined as

$$\vee(w_1, w_2, \ldots, w_L) = \frac{1}{L-1} \sum_{i=1}^{L} (i-1) w_i\,. \qquad (11)$$

The orness measure, also known as the attitudinal character of the aggregation, specifies the degree to which the OWA operator is like an *or* (*max*) operation. Specifically, when the degree of orness is greater than 0.5, the OWA operator is considered as more *orlike* than *andlike*. In this case, the larger arguments play a more important role in the aggregation process.

IV. Anomaly-based Android Malware Detection

In this section, we present DroidNMD, a novel technique that uses an ensemble classifier consisting of multiple one-class classifiers to detect Android malware by intercepting each application's incoming and outgoing traffic. DroidNMD consists of two main steps, namely, training and detection. In the following subsections, we describe each of these steps in detail.

A. Training Step

In this step, we take a set of Android benign applications and capture bidirectional flows for each application by running it on a real Android device. Recall that a flow is defined as a set of packets that share some common properties, such as source IP address, source port, destination IP address, destination port, and protocol. There are different types of flows [22]: unidirectional and bidirectional. A unidirectional flow, or briefly uniflow, is a flow composed only of packets sent from a single host to another single host. On the other hand, a bidirectional flow,

TABLE I
FEATURES CALCULATED FROM THE FLOWS OF AN APPLICATION AT EACH SLIDING WINDOW

Group	Description
Aggregation-based	The average time difference between consecutive flows
	The average number of bytes per flow
	The average number of packets per flow
	The average flow duration
	The maximum time difference between two consecutive flows
Entropy-based	The modified sample entropy of time differences between consecutive flows
	The modified sample entropy of the number of bytes per flow
	The modified sample entropy of the number of packets per flow
	The modified sample entropy of flow durations

or briefly biflow, is a flow composed of packets sent in both directions between two hosts. From now on, for the sake of convenience, we will refer to bidirectional flows simply as “flows”.

After capturing an application's flows, we extract a number of features from the flows using a sliding window technique. To do so, we use a sliding window of W consecutive time periods across the T total time periods, producing $T-W+1$ overlapping windows. For each window, we calculate the features shown in Table I. These features are divided into two different categories [19]: aggregation-based and entropy-based. Aggregation-based features measure the amount of network traffic consumed by the application and entropy-based features measure the degree of complexity (or predictability) in the application's network behavior. For each entropy-based feature, we set its values to –1 (i.e., predictable) if they are less than a cutoff threshold and to +1 (i.e., unpredictable) otherwise. We use another set of Android benign applications to determine the cutoff thresholds of all the entropy-based features. In doing so, we sort each feature's values in ascending order and then set the cutoff thresholds at the 2nd percentile of the values.

Subsequently, we create L training datasets $X_1, X_2, \dots, X_L$ of feature vectors by choosing L different overlapping subsets of features from the entire feature set and projecting all the applications' feature values on the feature subsets. Finally, we employ the training datasets to construct an ensemble classifier $\mathcal{C}$ consisting of L one-class classifiers $C_1, C_2, \dots, C_L$. For each one-class classifier $C_i \in \mathcal{C}$, the decision threshold, denoted by θ_i, is determined as

$$\theta_i : \frac{1}{|X_i|}\sum_{k=1}^{|X_i|} I(p_i(\mathbf{x}_{ik}|\omega_B) \geq \theta_i) = \nu_i \,, \tag{12}$$

where ω_B is the benign class, $\mathbf{x}_{ik}$ is a feature vector belonging to X_i, ν_i is the training acceptance rate of C_i, and I is an indicator function defined as

$$I(A) = \begin{cases} 1 & \text{if } A \text{ is true}\,, \\ 0 & \text{otherwise}\,. \end{cases} \tag{13}$$

B. Detection Step

In this step, we continuously capture flows for each unlabeled Android application and apply a sliding window technique to extract the features in Table I. Then, we create L different feature vectors per window, similar to that we described in the training step. Next, we give each window's feature vectors to the one-class classifiers of the ensemble classifier constructed in the training step. Subsequently, we reorder the outputs of the classifiers in ascending order and then aggregate them using a novel OWA operator, called NOWA, to predict the most likely class (“benign” or “suspicious”) the unlabeled application belongs to. Finally, we count the number of total windows in which the application has been classified as suspicious and report the application as malware if this number exceeds a predefined threshold. Formally, let $\mathbf{z}_1(t), \mathbf{z}_2(t), \dots, \mathbf{z}_L(t)$ be the feature vectors created in the current window t and $o_1(t), o_2(t), \dots, o_L(t) \in [0,1]$ be the outputs of the one-class classifiers $C_1, C_2, \dots, C_L$ such that $o_i(t) = p_i(\mathbf{z}_i(t)|\omega_B)$ for all $i = 1, 2, \dots, L$. Using the NOWA operator, the decision rule for the application is defined as

$$\ell(t) = \begin{cases} \omega_B & \text{if } \mathcal{O}_{\mathbf{w}}(o_1(t), o_2(t), \dots, o_L(t)) \geq \mathcal{O}_{\mathbf{w}}(\theta_1, \theta_2, \dots, \theta_L)\,, \\ \omega_S & \text{otherwise}\,, \end{cases} \tag{14}$$

where ω_S is the suspicious class, $\mathcal{O}_{\mathbf{w}}$ is the aggregation function given in (10), and $\mathbf{w}$ is the associated weighting vector of NOWA, whose elements w_i, for $i = 1, 2, \dots, L$, are defined as

$$w_i = \frac{2\eta_i}{3\eta_L + \eta_{L-1} - 1}\,, \tag{15}$$

where η_i is given by

$$\eta_i = 2\eta_{i-1} + \eta_{i-2}\,, \tag{16}$$

with $\eta_1 = 1$ and $\eta_2 = 2$. The application is reported as malware if the total number of suspicious windows among all previous windows exceeds a minimum detection threshold β:

$$\sum_{\tau=t_0}^{t} I(\ell(\tau) = \omega_S) \geq \beta\,. \tag{17}$$

In the following, we prove that the NOWA weights satisfy the condition of summing up to 1. In addition, we prove that the orness degree of NOWA is always in the range (0.5,1]. This causes it gives more importance to one-class classifiers that are more confident in their decision for the given feature vector.

Lemma 1. The NOWA weights satisfy the condition of summing up to 1.

PROOF. To prove this lemma, we compute

$$\begin{aligned} \sum_{i=1}^{L} w_i &= \sum_{i=1}^{L} \frac{2\eta_i}{3\eta_L + \eta_{L-1} - 1} \\ &= \frac{2}{3\eta_L + \eta_{L-1} - 1}\sum_{i=1}^{L} \eta_i\,. \end{aligned} \tag{18}$$

Now we need to compute $\sum_{i=1}^{L} \eta_i$. From (16), we have

$$\begin{aligned}\sum_{i=1}^{L} \eta_i &= \eta_1 + \eta_2 + \sum_{i=3}^{L} (2\eta_{i-1} + \eta_{i-2}) \\ &= 1 + 2\eta_1 + 2\sum_{i=3}^{L} \eta_{i-1} + \sum_{i=3}^{L} \eta_{i-2} \\ &= 1 + 2\eta_1 + 2\sum_{i=2}^{L-1} \eta_i + \sum_{i=1}^{L-2} \eta_i \\ &= 1 + 2\sum_{i=1}^{L} \eta_i - 2\eta_L + \sum_{i=1}^{L} \eta_i - \eta_L - \eta_{L-1} \\ &= 1 + 3\sum_{i=1}^{L} \eta_i - 3\eta_L - \eta_{L-1} \,.\end{aligned} \tag{19}$$

Thus, we get

$$\sum_{i=1}^{L} \eta_i = \frac{3\eta_L + \eta_{L-1} - 1}{2} \,. \tag{20}$$

By substituting (20) in (18), we find that $\sum_{i=1}^{L} w_i = 1$. □

Recall that orness is a measure of the degree of optimism. The larger the orness degree is, the more optimistic the aggregator is.

Lemma 2. The orness degree of NOWA is always in the range (0.5,1].

PROOF. It has already been shown that if the OWA weights are in ascending (or descending) order and the input arguments are sorted in ascending (or descending) order, then the degree of orness is always in the range (0.5,1] [23]. Thus, we only need to show the NOWA weights have the property that $w_i < w_{i+1}$ for $i = 1, 2, \dots, L-1$.

From (16), we know that $\eta_i \le \eta_{i+1}$ for $i = 1, 2, \dots, L-1$. Hence, for $i = 1, 2, \dots, L-1$, we have

$$\frac{2\eta_i}{3\eta_L + \eta_{L-1} - 1} \le \frac{2\eta_{i+1}}{3\eta_L + \eta_{L-1} - 1} \tag{21}$$

or

$$w_i \le w_{i+1} \,. \tag{22}$$

Therefore, we conclude that the NOWA weights are in ascending order. □

V. EXPERIMENTS

In this section, we evaluate the detection performance of DroidNMD by using a real dataset of Android benign applications and malware samples.

A. Dataset

The dataset used in our experiments consists of 180 and 97 Android benign applications and malware samples, respectively. The benign applications are selected among the most downloaded free applications on Google Play. We suppose the applications are not compromised, since they are daily downloaded by thousands of users and are frequently analyzed by security experts from around the world. The malware samples are randomly selected from Drebin [13]. We run each benign application and malware sample on a rooted Samsung Galaxy S3 running Android 4.4.4 KitKat and use the Android specific version of Tcpdump to capture packets.

We split our dataset into two parts: training and testing. The training dataset consists of 50 percent of the whole benign applications and is used to construct an ensemble classifier. The testing dataset consists of the remaining benign applications and all malware samples and is used exclusively for evaluation purposes.

A. Performance Measures

To evaluate the detection performance of DroidNMD, we use three standard measures: detection rate (DR), false alarm rate (FAR), and accuracy (ACC). DR is defined as the percentage of malware samples that are correctly classified, FAR is defined as the percentage of benign samples that are incorrectly classified, and ACC is defined as the percentage of benign and malware samples that are correctly classified. Since ACC considers both DR and FAR, we use it as the main performance measure in our experiments.

B. Experimental Setup

To construct an ensemble classifier, we use the traditional random subspace method (RSM) [24] with four simple heterogeneous one-class classifiers, namely, Gaussian (Gauss), K-nearest neighbor (KNN), Parzen-window (PW), and principal component analysis (PCA), and then build three models from each of them with the training acceptance rate of 97%. This results in an ensemble classifier of size $L = 12$. We assume that each random feature subset in RSM consists of 60% of the whole features in Table I.

Unless otherwise stated, we use the following parameter settings: the parameters of the sliding window technique are set to $W = 8$ and $T = 24$, resulting in 17 overlapping windows. The length of a time period is set to one hour. The parameters of mSampEn are set to $m = 2$ and $r = 0.003$. Finally, the threshold of the malware detection rule is set to $\beta = 4$.

All experiments are done in MATLAB using PRTools [25] and DDTools [26]. All reported results are the average of 30 independent runs. To compare different algorithms, we use the same feature subsets for their corresponding runs and then perform the Friedman test [27], with a significance level of 0.05, followed with the Nemenyi post-hoc test [27]. Recall that the Friedman test is a non-parametric statistical test often used for comparing more than two algorithms over multiple datasets. It ranks the algorithms for each dataset, and then checks to determine whether the measured average ranks are significantly different from the mean rank. The null hypothesis being tested is that there are no differences between the performances of all the algorithms. If the null hypothesis is rejected, then the Nemenyi post-hoc test is performed for pairwise comparisons. The results for the Nemenyi test are shown in critical difference diagrams, where the horizontal axis indicates the average ranks and the

TABLE II
PERFORMANCE MEASURES OF DROIDNMD FOR DIFFERENT VALUES OF β

Window	Average DR						Average FAR						Average ACC					
	$\beta=1$	$\beta=2$	$\beta=3$	$\beta=4$	$\beta=5$	$\beta=6$	$\beta=1$	$\beta=2$	$\beta=3$	$\beta=4$	$\beta=5$	$\beta=6$	$\beta=1$	$\beta=2$	$\beta=3$	$\beta=4$	$\beta=5$	$\beta=6$
1	83.13	0.00	0.00	0.00	0.00	0.00	0.11	0.00	0.00	0.00	0.00	0.00	91.19	48.13	48.13	48.13	48.13	48.13
2	89.86	72.58	0.00	0.00	0.00	0.00	1.33	0.04	0.00	0.00	0.00	0.00	94.10	85.76	48.13	48.13	48.13	48.13
3	92.51	80.41	67.39	0.00	0.00	0.00	1.33	0.07	0.00	0.00	0.00	0.00	95.47	89.80	83.08	48.13	48.13	48.13
4	96.56	87.35	78.28	66.19	0.00	0.00	1.37	0.19	0.04	0.00	0.00	0.00	97.56	93.35	88.72	82.46	48.13	48.13
5	97.84	92.51	83.92	75.26	64.05	0.00	1.59	0.33	0.15	0.04	0.00	0.00	98.11	95.95	91.59	87.15	81.35	48.13
6	98.97	96.70	89.31	83.78	73.09	57.01	2.81	0.41	0.19	0.15	0.04	0.00	98.11	98.09	94.37	91.52	86.02	77.70
7	98.97	97.90	95.53	89.21	82.54	69.04	4.07	1.59	0.26	0.19	0.15	0.04	97.50	98.15	97.56	94.31	90.87	83.92
8	100.00	97.90	97.77	94.47	87.01	78.38	5.52	1.74	1.33	0.19	0.15	0.04	97.34	98.07	98.20	97.04	93.19	88.77
9	100.00	98.97	97.77	97.63	91.31	84.71	6.78	1.93	1.41	0.22	0.15	0.04	96.74	98.54	98.16	98.66	95.42	92.05
10	100.00	99.90	98.93	97.73	95.43	89.90	6.85	1.93	1.41	0.22	0.15	0.04	96.70	99.02	98.77	98.72	97.56	94.74
11	100.00	99.90	99.86	98.76	96.56	94.16	6.89	1.93	1.41	0.22	0.15	0.04	96.68	99.02	99.25	99.25	98.15	96.95
12	100.00	100.00	99.86	99.73	98.66	95.29	7.19	1.96	1.41	0.22	0.15	0.04	96.54	99.06	99.25	99.75	99.23	97.54
13	100.00	100.00	99.97	99.76	99.62	97.42	8.41	2.30	1.41	0.22	0.15	0.04	95.95	98.89	99.30	99.77	99.73	98.65
14	100.00	100.00	100.00	99.90	99.69	98.56	8.41	2.30	1.41	0.22	0.15	0.04	95.95	98.89	99.32	99.84	99.77	99.23
15	100.00	100.00	100.00	99.97	99.76	98.59	8.70	2.30	1.41	0.22	0.15	0.04	95.81	98.89	99.32	99.88	99.80	99.25
16	100.00	100.00	100.00	99.97	99.79	98.63	9.85	2.59	1.41	0.22	0.15	0.04	95.26	98.75	99.32	99.88	99.82	99.27
17	100.00	100.00	100.00	99.97	99.86	99.76	10.00	2.59	1.44	0.22	0.15	0.04	95.19	98.75	99.30	99.88	99.86	99.86

TABLE III
COMPARISON OF NOWA WITH THREE COMBINATION RULES IN TERMS OF THE AVERAGE DR, FAR, AND ACC

Window	Average DR				Average FAR				Average ACC			
	NOWA	**Majority**	**Mean**	**SOWA**	**NOWA**	**Majority**	**Mean**	**SOWA**	**NOWA**	**Majority**	**Mean**	**SOWA**
1	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	48.13	48.13	48.13	48.13
2	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	48.13	48.13	48.13	48.13
3	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	48.13	48.13	48.13	48.13
4	66.19	61.62	63.99	64.40	0.00	0.00	0.00	0.04	82.46	80.09	81.32	81.52
5	75.26	71.55	73.54	73.99	0.04	0.07	0.11	0.07	87.15	85.20	86.22	86.47
6	83.78	80.82	82.27	82.68	0.15	0.07	0.22	0.19	91.52	90.02	90.70	90.93
7	89.21	86.29	87.87	88.11	0.19	0.11	0.22	0.19	94.31	92.83	93.60	93.74
8	94.47	92.23	93.47	93.75	0.19	0.11	0.22	0.19	97.04	95.92	96.51	96.67
9	97.63	95.36	96.70	96.94	0.22	0.22	0.37	0.26	98.66	97.49	98.11	98.29
10	97.73	96.19	97.04	97.32	0.22	0.22	0.37	0.26	98.72	97.91	98.29	98.48
11	98.76	97.56	98.18	98.42	0.22	0.22	0.37	0.26	99.25	98.63	98.88	99.06
12	99.73	98.56	99.21	99.35	0.22	0.22	0.37	0.26	99.75	99.14	99.41	99.54
13	99.76	98.59	99.21	99.35	0.22	0.22	0.37	0.26	99.77	99.16	99.41	99.54
14	99.90	99.14	99.52	99.59	0.22	0.22	0.37	0.26	99.84	99.45	99.57	99.66
15	99.97	99.52	99.76	99.79	0.22	0.22	0.37	0.26	99.88	99.64	99.70	99.77
16	99.97	99.52	99.79	99.79	0.22	0.22	0.37	0.26	99.88	99.64	99.71	99.77
17	99.97	99.52	99.79	99.79	0.22	0.33	0.41	0.26	99.88	99.59	99.70	99.77

TABLE IV
RESULTS OF THE FRIEDMAN TEST FOR RANKING THE COMBINATION RULES USING ACC AS THE PERFORMANCE MEASURE

Statistical Test			Average Ranks			
Significance Level	**P-value**	**Significant Differences**	**NOWA**	**Majority**	**Mean**	**SOWA**
0.05	2.35×10^{-46}	Yes	2.10	2.80	2.64	2.46

groups of algorithms that are not significantly different are connected with a colored line on the top of the axis.

C. Experimental Results

We run experiments to analyze how different parameters and operators affect the detection performance of DroidNMD.

In the first set of experiments, we investigate the usefulness of the history parameter, β. Table II reports the detection performance of DroidNMD in terms of the average DR, FAR, and ACC for different values of β over 17 overlapping windows. From the table, we observe that by increasing the value of β up to 4, we obtain a slightly lower value of DR and a significantly

TABLE V
COMPARISON OF DROIDNMD WITH FOUR SINGLE ONE-CLASS CLASSIFIERS IN TERMS OF THE AVERAGE DR, FAR, AND ACC

Window	Average DR					Average FAR					Average ACC				
	DroidNMD	**Gauss**	**KNN**	**PW**	**PCA**	**DroidNMD**	**Gauss**	**KNN**	**PW**	**PCA**	**DroidNMD**	**Gauss**	**KNN**	**PW**	**PCA**
1	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	48.13	48.13	48.13	48.13	48.13
2	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	48.13	48.13	48.13	48.13	48.13
3	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	48.13	48.13	48.13	48.13	48.13
4	66.19	44.33	69.07	69.07	67.01	0.00	0.00	1.11	0.00	0.00	82.46	71.12	83.42	83.96	82.89
5	75.26	56.70	77.32	77.32	76.29	0.04	0.00	2.22	1.11	0.00	87.15	77.54	87.17	87.70	87.70
6	83.78	68.04	85.57	85.57	85.57	0.15	0.00	3.33	1.11	0.00	91.52	83.42	90.91	91.98	92.51
7	89.21	72.16	91.75	90.72	89.69	0.19	0.00	3.33	1.11	0.00	94.31	85.56	94.12	94.65	94.65
8	94.47	82.47	94.85	94.85	93.81	0.19	0.00	4.44	1.11	0.00	97.04	90.91	95.19	96.79	96.79
9	97.63	85.57	97.94	97.94	96.91	0.22	1.11	5.56	2.22	1.11	98.66	91.98	96.26	97.86	97.86
10	97.73	89.69	97.94	97.94	97.94	0.22	1.11	5.56	2.22	1.11	98.72	94.12	96.26	97.86	98.40
11	98.76	92.78	98.97	98.97	98.97	0.22	1.11	5.56	2.22	1.11	99.25	95.72	96.79	98.40	98.93
12	99.73	93.81	100.00	100.00	100.00	0.22	1.11	6.67	2.22	1.11	99.75	96.26	96.79	98.93	99.47
13	99.76	93.81	100.00	100.00	100.00	0.22	1.11	6.67	2.22	1.11	99.77	96.26	96.79	98.93	99.47
14	99.90	96.91	100.00	100.00	100.00	0.22	1.11	6.67	2.22	1.11	99.84	97.86	96.79	98.93	99.47
15	99.97	97.94	100.00	100.00	100.00	0.22	1.11	7.78	2.22	1.11	99.88	98.40	96.26	98.93	99.47
16	99.97	97.94	100.00	100.00	100.00	0.22	1.11	7.78	2.22	2.22	99.88	98.40	96.26	98.93	98.93
17	99.97	97.94	100.00	100.00	100.00	0.22	2.22	7.78	2.22	2.22	99.88	97.86	96.26	98.93	98.93

TABLE VI
RESULTS OF THE FRIEDMAN TEST FOR RANKING THE ONE-CLASS CLASSIFIERS USING ACC AS THE PERFORMANCE MEASURE

Statistical Test			Average Ranks				
Significance Level	**P-value**	**Significant Differences**	**DroidNMD**	**Gauss**	**KNN**	**PW**	**PCA**
0.05	4.05×10^{-274}	Yes	1.88	4.41	3.92	2.55	2.24

lower value of FAR, resulting in a higher value of ACC. This may be due to the fact that by increasing the value of β, DroidNMD reports an application as malware if it observes more suspicious windows in past in order to prevent making hasty decisions, which causes the decrease in the number of false positives is much more than the decrease in the number of true positives.

In the second set of experiments, we evaluate the overall impact of NOWA on the detection performance of DroidNMD. For this purpose, we use two popular combination rules, namely Majority and Mean, and a simple OWA operator, proposed in [28], to aggregate the outputs of one-class classifiers in the ensemble classifier constructed by DroidNMD. For convenience, we refer to this simple OWA operator as SOWA. Table III compares NOWA with these combination rules in terms of the average DR, FAR, and ACC. Obviously, NOWA has the best detection performance among all the combination rules. We assign ranks to the corresponding runs of the combination rules, using ACC as the performance measure. The best combination rule is assigned the rank of 1. We apply the Friedman test to determine whether the measured average ranks are significantly different from the mean rank, which is 2.5 in our case, under the null hypothesis. The results, as given in Table IV, show that the null hypothesis is rejected at the significance level of 0.05. Thus, we apply the Nemenyi post-hoc test for pairwise comparisons. Fig. 2 shows the critical difference diagram for the Nemenyi test. A significant difference between two combination rules occurs when the difference in rankings is greater than the critical difference, which is 0.20 in this test. From the figure, we notice that NOWA has the best average rank, 2.10, which is significantly better than that of Majority, Mean, and SOWA.

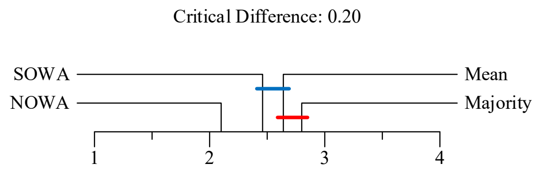

Fig. 2. Pairwise comparison of the combination rules using the Nemenyi test.

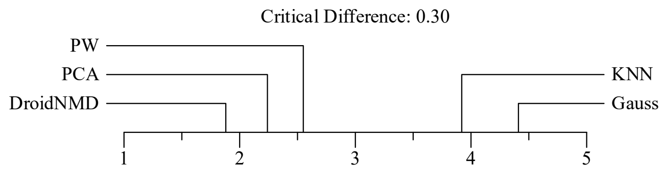

Fig. 3. Pairwise comparison of the one-class classifiers using the Nemenyi test.

In the third set of experiments, we compare the detection performance of DroidNMD with that of four single one-class classifiers, namely, Gauss, KNN, PW, and PCA. Table V reports the obtained results. Obviously, DroidNMD has a much lower FAR while having a comparable DR, resulting in a higher ACC. Similar to the previous experiment, we perform the Friedman test followed with the Nemenyi test to validate the statistical significance of ranking differences for pairwise comparisons.

Table VI and Fig. 3 show the obtained results. Clearly, we notice that DroidNMD has the best average rank, 1.88, which is significantly better than that of Gauss, KNN, PW, and PCA. Therefore, we conclude that ensemble classifiers can be considered as an effective tool for anomaly malware detection in Andriod.

VI. Conclusion

Given the widespread growth of malware threats on Android, there is an urgent need for techniques that can effectively combat these threats. To meet this need, researchers have proposed various misuse and anomaly detection techniques. Many of these techniques show promising results, but they often fail to cope with zero-day malware or produce many false alarms. In this paper, we have addressed these shortcomings by presenting DroidNMD, a novel network-based anomaly detection technique that applies ensemble classification to improve the accuracy of malware detection in Android. DroidNMD constructs an ensemble classifier consisting of multiple one-class classifiers and aggregates the outputs of the classifiers using a novel OWA operator, called NOWA.

We have conducted three experiments to analyze how different parameters and operators affect the detection performance of DroidNMD. In particular, we have evaluated the overall impact of NOWA on the detection performance of DroidNMD. Furthermore, we have compared the detection performance of DroidNMD with that of four different single one-class classifiers, namely, Gauss, KNN, PW, and PCA. The experimental results have shown that NOWA outperforms popular combination rules in terms of accuracy. In addition, DroidNMD significantly increases the accuracy as compared to single one-class classifiers. Consequently, DroidNMD classifies Android applications as either benign or malware, with a high detection rate close to 100% and a low false alarm rate close to 0.20%.

References

[1] Statista. (2017). *Number of available applications in the Google Play store from December 2009 to March 2017*. [Online]. Available: https://www.statista.com/statistics/266210

[2] Statista. (2017). *Installed base of smartphones by operating system from 2015 to 2016*. [Online]. Available: https://www.statista.com/statistics/385001

[3] C. Lueg. (2017). *8,400 new Android malware samples every day*. [Online]. Available: https://www.gdatasoftware.com/blog

[4] P. Faruki, A. Bharmal, V. Laxmi, V. Ganmoor, M. S. Gaur, M. Conti, and M. Rajarajan, “Android security: A survey of issues, malware penetration, and defenses,” *IEEE Commun. Surveys Tuts.*, vol. 17, no. 2, pp. 998–1022, Q2 2015.

[5] C.-Y. Huang, Y.-T. Tsai, and C.-H. Hsu, “Performance evaluation on permission-based detection for Android malware,” in *Advances in Intelligent Systems and Applications*, J.-S. Pan, C.-N. Yang, and C.-C. Lin, Eds. Berlin, Heidelberg, Germany: Springer, 2013, pp. 111–120.

[6] V. Rastogi, Y. Chen, and X. Jiang, “DroidChameleon: Evaluating Android anti-malware against transformation attacks,” in *Proc. 8th ACM SIGSAC Symp. Inform., Comput. Commun. Secur. (ASIA CCS '13)*, Hangzhou, China, 2013, pp. 329–334.

[7] Y. Xue, G. Meng, Y. Liu, T. H. Tan, H. Chen, J. Sun, and J. Zhang, “Auditing anti-malware tools by evolving Android malware and dynamic loading technique,” *IEEE Trans. Inf. Forensics Secur.*, vol. 12, no. 7, pp. 1529–1544, Jul. 2017.

[8] C.-Y. Huang, C.-H. Chiu, C.-H. Lin, and H.-W. Tzeng, “Code coverage measurement for Android dynamic analysis tools,” in *Proc. 2015 IEEE Int. Conf. Mobile Services (MS '15)*, New York, NY, USA, 2015. pp. 209–216.

[9] E. Parhizkar and M. Abadi, “BeeOWA: A novel approach based on ABC algorithm and induced OWA operators for constructing one-class classifier ensembles,” *Neurocomputing*, vol. 166, pp. 367–381, Oct. 2015.

[10] R. R. Yager, “On ordered weighted averaging aggregation operators in multicriteria decision making,” *IEEE Trans. Syst., Man, Cybern.*, vol. 18, no. 1, pp. 183–190, Jan. 1988.

[11] E. Parhizkar and M. Abadi, “OC-WAD: A one-class classifier ensemble approach for anomaly detection in web traffic,” in *Proc. 2015 23rd Iranian Conf. Elect. Eng. (ICEE '15)*, Tehran, Iran, 2015, pp. 631–636.

[12] Y. Aafer, W. Du, and H. Yin, “DroidAPIMiner: Mining API-level features for robust malware detection in Android,” in *Security and Privacy in Communication Networks*, T. Zia, A. Zomaya, V. Varadharajan, and M. Mao, Eds. Cham, Switzerland: Springer International Publishing, 2013, pp. 86–103.

[13] D. Arp, M. Spreitzenbarth, M. Hübner, H. Gascon, and K. Rieck, “Drebin: Effective and explainable detection of Android malware in your pocket,” in *Proc. 2014 Network Distributed Syst. Secur. Symp. (NDSS '14)*, San Diego, CA, USA, 2014, pp. 1–12.

[14] Y. Feng, S. Anand, I. Dillig, and A. Aiken, “Apposcopy: Semantics-based detection of Android malware through static analysis,” in *Proc. 22nd ACM SIGSOFT Int. Symp. Found. Softw. Eng. (FSE '14)*, Hong Kong, China, 2014, pp. 576–587.

[15] W. Yang, X. Xiao, B. Andow, S. Li, T. Xie, and W. Enck, “AppContext: Differentiating malicious and benign mobile app behaviors using context,” in *Proc. 2015 37th IEEE Int. Conf. Softw. Eng. (ICSE '15)*, vol. 1, Florence, Italy, 2015, pp. 303–313.

[16] S. K. Dash, G. Suarez-Tangil, S. Khan, K. Tam, M. Ahmadi, J. Kinder, and L. Cavallaro, “DroidScribe: Classifying Android malware based on runtime behavior,” in *Proc. 2016 IEEE Secur. Privacy Workshops (SPW '16)*, San Jose, CA, USA, 2016, pp. 252–261.

[17] W. Wang, Y. Li, X. Wang, J. Liu, and X. Zhang, “Detecting Android malicious apps and categorizing benign apps with ensemble of classifiers,” *Future Gener. Comput. Syst.*, vol. 78, no. 3, pp. 987–994, Jan. 2018.

[18] A. Shabtai, L. Tenenboim-Chekina, D. Mimran, L. Rokach, B. Shapira, and Y. Elovici, “Mobile malware detection through analysis of deviations in application network behavior,” *Comput. Secur.*, vol. 43, pp. 1–18, Jun. 2014.

[19] F. Ghaffari and M. Abadi, “DroidMalHunter: A novel entropy-based anomaly detection system to detect malicious Android applications,” in *Proc. 2015 5th Int. Conf. Comput. Knowl. Eng. (ICCKE '15)*, Mashhad, Iran, 2015, pp. 301–306.

[20] A. Saracino, D. Sgandurra, G. Dini, and F. Martinelli, “MADAM: Effective and efficient behavior-based Android malware detection and prevention,” *IEEE Trans. Dependable Secure Comput.*, to be published.

[21] H.-B. Xie, W.-X. He, and H. Liu, “Measuring time series regularity using nonlinear similarity-based sample entropy,” *Phys. Lett. A*, vol. 372, no. 48, pp. 7140–7146, Dec. 2008.

[22] B. H. Trammell and E. Boschi, “RFC 5103: Bidirectional flow export using IP flow information export (IPFIX),” Internet Engineering Task Force (IETF), Tech. Rep., 2008.

[23] D. Filev, R. R. Yager, “Analytic properties of maximum entropy OWA operators,” *Inf. Sci.*, vol. 85, no. 1–3, pp. 11–27, Jul. 1995.

[24] T. K. Ho, “The random subspace method for constructing decision forests,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 20, no. 8, pp. 832–844, Aug. 1998.

[25] D. de Ridder, D. M. J. Tax, B. Lei, G. Xu, M. Feng, Y. Zou, F. van der Heijden, *Classification, Parameter Estimation and State Estimation: An Engineering Approach Using MATLAB*, 2nd ed. Hoboken, NJ, USA: John Wiley & Sons, 2017.

[26] D. M. J. Tax. (2015). *DDTools, the data description toolbox for MATLAB*, version 2.1.2. [Online]. Available: http://prlab.tudelft.nl/david-tax/dd_tools.html

[27] M. Hollander, D. A. Wolfe, and E. Chicken, *Nonparametric Statistical Methods*, 3rd ed. Hoboken, NJ, USA: John Wiley & Sons, 2014.

[28] R. R. Yager, “Families of OWA operators,” *Fuzzy Sets Syst.*, vol. 59, no. 2, pp. 125–148, Oct. 1993.