Anatomizing Android Malwares

1st Anand Tirkey
Computer Science and Engineering
National Institute of Technology
Rourkela, India
andy9c@gmail.com

2nd Ramesh Kumar Mohapatra
Computer Science and Engineering
National Institute of Technology
Rourkela, India
rkmohapatra@ieee.org

3rd Lov Kumar
Computer Science and Information Systems
BITS Pilani, Hyderabad Campus
Hyderabad, India
lovkumar505@gmail.com

***Abstract*—Android OS being the popular choice of majority users also faces the constant risk of breach of confidentiality, integrity and availability (CIA). Effective mitigation efforts needs to identified in order to protect and uphold the CIA triad model, within the android ecosystem. In this paper, we propose a novel method of android malware classification using Object-Oriented Software Metrics and machine learning algorithms. First, android apps are decompiled and Object-Oriented Metrics are obtained. VirusShare service is used to tag an app either as malware or benign. Object-Oriented Metrics and malware tag are clubbed together into a dataset. Eighty different machine-learned models are trained over five thousand seven hundred and seventy four android apps. We evaluate the performance and stability of these models using it's malware classification accuracy and AUC (area under ROC curve) values. Our method yields an accuracy and AUC of 99.83% and 1.0 respectively.**

***Index Terms*—android, malware detection, machine learning, object-oriented metrics**

I. Introduction

Since its unveiling in 2007, Android OS market continues to grow at an unprecedented rate and is expected to grab global market share at 86.7% by 2019 according to International Data Corporation, USA. Meanwhile, Symantec internet security threat report 2019 addressed that in 2018, it intercepted & blocked an average of 10,573 malicious mobile apps per day. In May 2019, Google Inc. announced that 42.1% of the total android mobile users run on unsupported versions of the OS. Karstern Nohl *et al*. [1] have alarmingly exhorted that major mobile hardware vendors have inculcated the dangerous practice of skipping monthly android patches without the end-user's knowledge. In practice, either the hardware vendor stops providing regular android updates or the android system version in manipulated to emulate successful android update installation. These issues have further worsened the ever-widening gap in android os fragmentation as shown in Table I.

Google has put forward various preventive measures in order to prevent malware attack such as improved android permission and google play protect app. Recent android os asks the user for permission only when the app truly requires it as compared to old versions of os that required all permissions during installation. Even though this method prevents suspicious malware activities to some extent, it becomes a usual habit for the users in this scenario to grant permissions every time any app asks, without having a proper understanding of the consequences of granting the permission. It is also possible to install android apps from third party sources other than the official google play app, hence this preventive measure also falls short of being an effective protection between end-user and malware.

TABLE I: Android OS Fragmentation 2019

Android OS Fragmentation 2019				
Version	**Codename**	**API**	**Distribution**	**Status**
2.3.3 -2.3.7	Gingerbread	10	0.3%	Unsupported
4.0.3 -4.0.4	Ice Cream Sandwich	15	0.3%	
4.1.x	Jelly Bean	16	1.2%	
4.2.x		17	1.5%	
4.3		18	0.5%	
4.4	KitKat	19	6.9%	
5.0	Lollipop	21	3.0%	
5.1		22	11.5%	
6.0	Marshmallow	23	16.9%	
7.0	Nougat	24	11.4%	Supported
7.1		25	7.8%	
8.0	Oreo	26	12.9%	
8.1		27	15.4%	
9.0	Pie	28	10.4%	

II. Preliminaries

A. Android APK

Android Package (APK) is an executable file under android ecosystem. Basically it is a zip file that encapsulates all the resources required for an android app to run. APK file primarily consists of "classes.dex" (dalvik executable file) that stores all the compiled source codes. The APK archive may contain more than one "classes.dex" file such as "classes1.dex", "classes2.dex" etc. All other resources such as images, audio, database etc. are stored inside APK archive.

B. Android APK Decompilation

Since "classes.dex" inside APK archive contains the compiled source codes, it becomes the starting point for

TABLE II: List of selected Object-Oriented Metrics

Sl.	Abbreviation	Description
1	WMC	Weighted methods per class
2	DIT	Depth of Inheritance Tree
3	NOC	Number of Children
4	CBO	Coupling between object classes
5	RFC	Response for a Class
6	LCOM	Lack of cohesion in methods
7	Ca	Afferent coupling
8	Ce	Efferent coupling
9	NPM	Number of Public Methods for a class
10	LCOM3	Lack of cohesion in methods Henderson-Sellers version
11	LCO	Lines of Code
12	DAM	Data Access Metric
13	MOA	Measure of Aggregation
14	MFA	Measure of Functional Abstraction
15	CAM	Cohesion Among Methods of Class
16	IC	Inheritance Coupling
17	CBM	Coupling Between Methods
18	AMC	Average Method Complexity

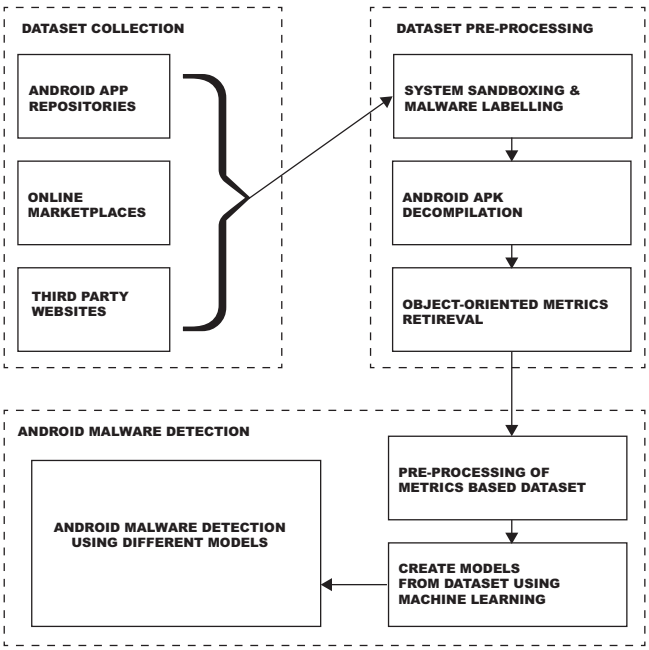


Fig. 1: Android Malware Detection Model

android decompilation process. First, "classes.dex" file is extracted and then apk decompilation tools such as dex2jar [2] is used to decompile android apk into a jar (java archive file). This jar file is then used to retrieve the Object-Oriented Software metrics upon further processing.

C. Object-Oriented Metrics

In this paper, 18 Object-Oriented Software metrics have been used as shown in Table II. These metrics are defined by several authors such as Chidamber *et al*. [3], Martin *et al*. [4], Bansiya *et al*. [5], Henderson *et al*. [6], Halstead *et al*. [7] and Tang *et al*. [8].

Since android apps are primarily programmed using java programming language, that uses Object-Oriented programming paradigm. It natively becomes possible to obtain Object-Oriented metrics for android apps. These metrics can be used as features and also can be instrumental in classifying android malware.

III. Related Work

Over the years, numerous techniques have been used for malware classification which can primarily be categorised into static analysis and dynamic analysis. Various authors have used static analysis such as Zhuo Ma *et al*. [9] make a structured control-flow graph of the android app from which API information is extracted. Three different datasets are obtained from this information *i.e*., API calls, API frequency and API sequence datasets. These datasets are then used over an ensemble technique deploying C4.5, DNN and LSTM algorithms. Neeraj Chavan *et al*. [10] collect the list of permissions specifically requested by android apps as their features for malware classification using Random forest, ANN, SVM, AdaBoost etc. Shivi Garg *et al*. [11] include both static features and dynamic features such as API calls, permissions, battery temperature, network traffic etc. amongst others. Classification techniques such as MLP, SVM, PART, RIDOR and their ensemble are used for malware detection. Yao-Saint Yen *et al*. [12] generate images from code, first by extracting important words from every apk using TF-IDF (term frequency inverse-document frequency), then they deploy SimHash and gjb2 algorithm to generate images from the collection of important words. Finally, CNN is used for malware classification. Alejandro Martín *et al*. [13] have used dynamic analysis features obtained from DroidBox tool. These features are then modelled into first-order markov chains. The transition probabilities and state-frequencies obtained from this first-order markov model acts as input for deep-learning algorithms such as CNN, RNN, LSTM etc. for detecting android malware. Dina Saif *et al*. [14] use both static analysis and dynamic analysis features along with system calls in their dataset. They obtain static information using Rapid Android Parser jar for Investigating DEX (RAPID) tool and DynaLog tool is used to obtain dynamic runtime information. All the information is clubbed together in a common dataset which uses relief feature selection method for reducing dimensionality. Finally, Deep Belief Networks (DBN) are used for malware detection. Ignacio Martín *et al*. [15] have collected android apps and have tagged the apps either as benign or malware. The app is flagged as malware if atleast one of the 61 antivirus softwares they have used successfully detects it as a malware otherwise it si labelled as benign. They have also used graph-community algorithms and hierarchical clustering

algorithms to club unsuspecting malware apps. Finally, Logistic regression and Random Forest techniques are used for malware classification. Xi Xiao *et al*. [16] build two classification models based on LSTM using system call sequences as their features. One of these models is trained with system call sequence from malware apps and the other is trained with system call sequence from benign apps. Whenever an incoming android app system call sequence is encountered, that sequence is run against both of these models and a similarity score is generated for each of the models. If the similarity score of the new sequence against the malware model is greater than that of the benign model, then the incoming app is classified as a malware, otherwise it is classified as benign. Wei Wang *et al*. [17] obtain their features using Androguard [18] tool. They deploy Deep auto-encoder (DAE) and CNN model for malware classification. Firstly, they apply multiple CNN over high-dimensional features in order to detect android malware. Consequently, the CNN-S (Serial Convolution Neural Network) uses Relu activation function to increase sparseness dropout in order to prevent over-fitting. Finally, DAE is applied over CNN as pre-training of CNN for reducing training time. This model is then used for effective malware classification. Hui-Juan Zhu *et al*. [19] use ensemble machine learning algorithms (Random Forest, SVM etc.) for detecting android malwares. They selected commonly available group of features such as permissions, monitoring system events, sensitive API and permission rate for the collected android apps. The ensemble of classifiers is fine-tuned using two parameters *i.e.*, the selection of features at each split and the number of decision trees.

IV. Proposed Work

The proposed work in categorized into three primary components such as effectiveness of Object-Oriented metrics, metrics-based dataset preparation followed by dataset pre-processing and malware classification.

A. Effectiveness of Object-Oriented Metrics

It is imperative to establish that Object-Oriented Metrics are feasible and effective in classifying android malware. Figure 2 shows the box-plot of Object-Oriented Metrics as described in Table II. Each subfigure from Figure 2 has two box-plots that shows the distribution of Object-Oriented metric values both for benign and malware apps. It is observed that the IQR (interquartile range) for most of the Object-Oriented metrics don't overlap and are distinct except for Ca & AMC metrics as shown in Figure 2(g) & Figure 2(r) respectively. Therefore, because of the multiple existence of non-overlapping distinct IQR of malware & benign apps, these Object-Oriented metrics can be used in classifying malware apps.

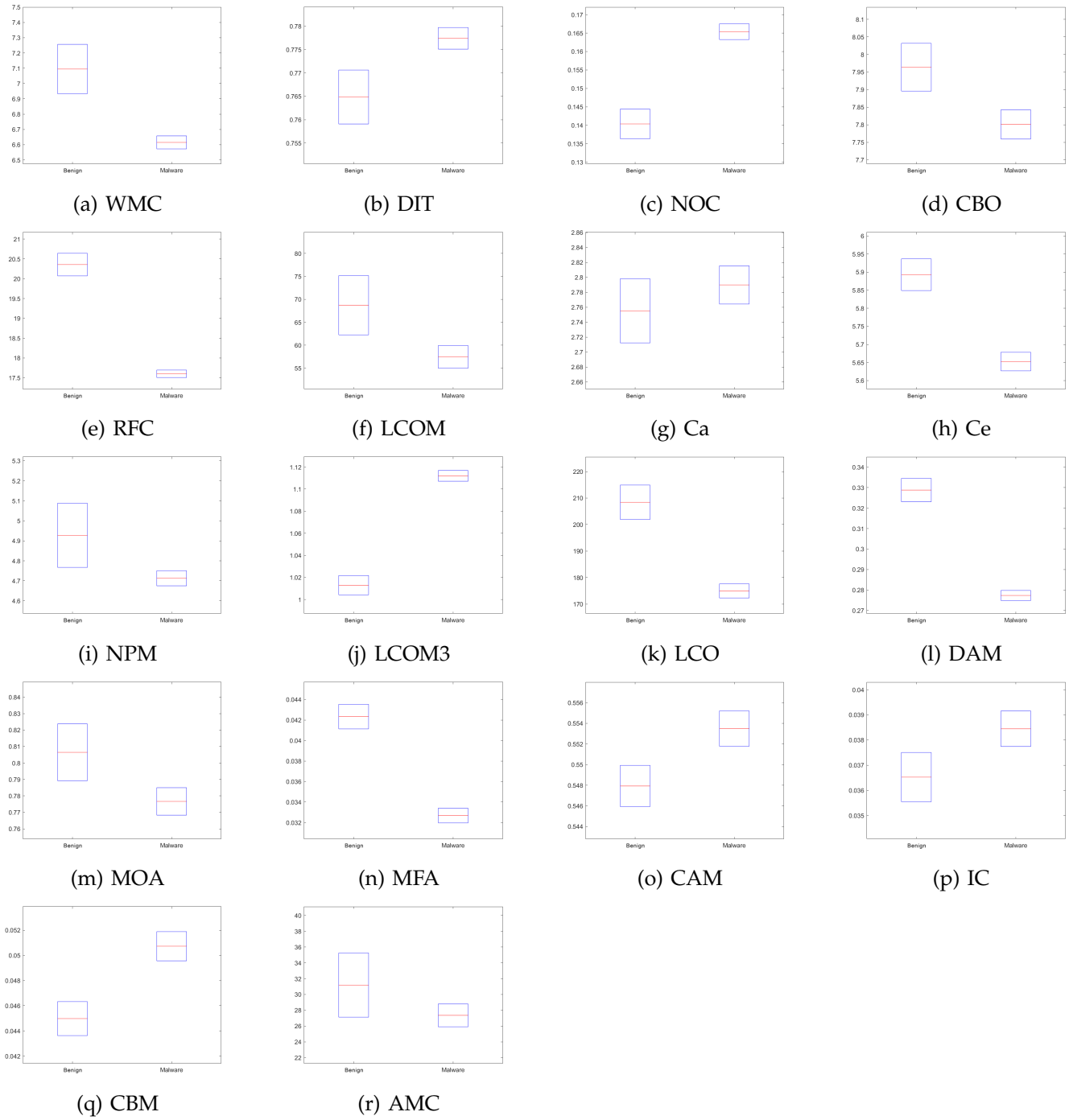
$$IQR = (Q3 - Q1),\ where$$
$$Q3 \rightarrow Third\ Quartile \quad (1)$$
$$Q1 \rightarrow First\ Quartile$$

B. Metrics-based dataset preparation

The proposed android malware detection model as shown in Figure 1 is grouped into three phases such as dataset collection phase, dataset pre-processing phase and android malware detection phase. Each of these phases further comprise of sub-processes. In data collection phase, android apps are collected from android app repositories, online marketplaces and from third-party website sources. These collected apps are stored in a local repository. In data pre-processing phase, the apps from local repository are decompiled using dex2jar [2] and it's Object-Oriented metrics are obtained using CKJM-extended tool [20]. An android app is determined to be benign or malware using VirusTotal service. VirusTotal is supported by Google Inc. and is a community based platform that keeps track whether an app is benign or malware. Finally, for each android app a feature vector of dimension (1×20) is designated as shown in Table III. The metrics-based dataset dimension is ($N \times 20$), where N is the total number of android apps considered for the experiment. In this dataset, the first feature is app name followed by 18 Object-Oriented Metrics as shown in Table II and a malware tag (benign or malware). This metrics-based dataset forms the basis of malware classification using machine learning techniques.

C. Dataset pre-processing & malware classification

A total of five thousand seven hundred and seventy four android apps are considered for this experiment. Out of which one thousand five hundred and eighty two are malware and the remaining are benign apps. It is observed that the malware apps are comparatively less than benign ones. Hence, SMOTE [21] (Synthetic Minority Oversampling Technique) analysis is chosen for producing class-balanced dataset. Henceforth, two different datasets *i.e.*, original dataset (OD) and class-balanced smote dataset (SMOTE) will be used for evaluation in our experiments. Further, four feature selection techniques are considered such as including all features without any exclusion (ALL), wilcoxon signed-rank test (SIG), univariate logistic regression (ULR) and principal component analysis (PCA). Finally, ten different machine learning algorithms such as Logistic Regression (LOGR), Decision Tree (DT), Artificial neural network with adaptive normalised nonlinear gradient descent (ANNGD) [22], Artificial neural network with gradient descent-momentum (ANNGDM) [23], Artificial neural network using gradient descent with momentum and adaptive learning rate backpropagation (ANNGDMX)



(a) WMC (b) DIT (c) NOC (d) CBO

(e) RFC (f) LCOM (g) Ca (h) Ce

(i) NPM (j) LCOM3 (k) LCO (l) DAM

(m) MOA (n) MFA (o) CAM (p) IC

(q) CBM (r) AMC

Fig. 2: Object-Oriented Metrics Boxplot

[24], Least squared support vector machine (LSSVM) [25] with linear kernel (LSSVM-LIN), polynomial kernel (LSSVM-PLY) & radial basis function kernel (LSSVM-RBF), Bagged tree ensemble (BTE) [26] and Majority voting ensemble (MVE) [27] are used to create ten different classification models, whose discriminatory power is assessed using two parameters *i.e.*, classifier's malware detection accuracy and classifier's AUC (area under the curve) value.

V. Experimental Results & Comparison

In our experiment, additional six different datasets are obtained from OD & SMOTE dataset *i.e.*, OD-SIG dataset (dataset obtained by applying SIG over OD), SMOTE-SIG dataset (dataset obtained by applying SIG over SMOTE), OD-ULR dataset (dataset obtained by applying ULR over OD), SMOTE-ULR dataset (dataset

TABLE III: Object-Oriented metrics dataset feature vector

Metrics-Based Dataset Feature Vector										
app_name	**WMC**	**DIT**	**NOC**	**CBO**	**RFC**	**LCOM**	**Ca**	**Ce**	**NPM**	**...**
...	**LCOM3**	**LCO**	**DAM**	**MOA**	**MFA**	**CAM**	**IC**	**CBM**	**AMC**	**malware_tag**

TABLE IV: Classifier Accuracy against different datasets applying various feature selection techniques

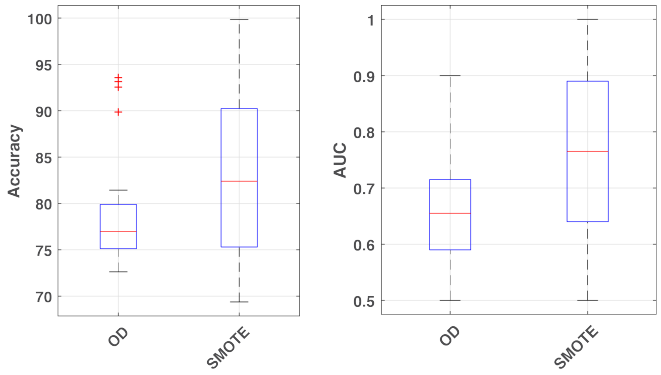
	LOGR	DT	ANNGD	ANNGDM	ANNGDMX	LSSVM-LIN	LSSVM-PLY	LSSVM-RBF	BTE	MVE
OD	79.38	77.90	76.08	72.62	76.43	79.46	93.15	79.91	77.90	80.68
SMOTE	82.32	88.62	74.64	70.70	72.56	74.09	99.61	82.48	99.61	89.99
OD-SIG	81.44	76.86	75.20	72.62	74.26	79.18	89.86	81.27	76.86	79.88
SMOTE-SIG	82.27	87.96	75.97	78.84	74.59	89.28	**99.83**	82.15	99.83	90.49
OD-ULR	78.61	76.32	76.00	72.62	76.17	80.16	93.58	79.05	76.32	79.55
SMOTE-ULR	81.60	87.13	77.28	69.89	74.64	89.61	99.23	81.49	99.23	91.82
OD-PCA	76.78	75.04	73.57	72.96	72.70	80.59	92.55	77.12	75.04	74.35
SMOTE-PCA	79.96	83.60	70.06	70.40	69.38	85.04	99.72	76.75	99.72	86.53

TABLE V: Classifier AUC against different datasets applying various feature selection techniques

	LOGR	DT	ANNGD	ANNGDM	ANNGDMX	LSSVM-LIN	LSSVM-PLY	LSSVM-RBF	BTE	MVE
OD	0.68	0.73	0.59	0.50	0.62	0.65	0.89	0.68	0.73	0.66
SMOTE	0.76	0.86	0.60	0.53	0.59	0.58	1.00	0.75	1.00	0.87
OD-SIG	0.71	0.72	0.58	0.50	0.58	0.64	0.84	0.70	0.72	0.64
SMOTE-SIG	0.77	0.85	0.63	0.69	0.60	0.86	**1.00**	0.75	1.00	0.91
OD-ULR	0.67	0.72	0.59	0.50	0.60	0.67	0.90	0.65	0.72	0.64
SMOTE-ULR	0.76	0.85	0.66	0.50	0.65	0.86	0.99	0.74	0.99	0.91
OD-PCA	0.62	0.68	0.52	0.51	0.51	0.67	0.87	0.60	0.68	0.53
SMOTE-PCA	0.73	0.81	0.51	0.52	0.53	0.80	1.00	0.66	1.00	0.81

obtained by applying ULR over SMOTE), OD-PCA dataset (dataset obtained by applying PCA over OD) and SMOTE-PCA (dataset obtained by applying PCA over SMOTE). These eight datasets are used to create models for each of the classification algorithms and it's accuracy & AUC is shown in Table IV & table V respectively. It is observed that LSSVM-PLY over SMOTE-SIG dataset provides a better malware detection accuracy of 99.83% and AUC of 1.0.

In order to assess the performance of both OD and SMOTE datasets, suitable box-plot has been illustrated in Figure 3. It is observed from Figure 3(a) and Figure 3(b) that, a class-balanced SMOTE dataset generally provides better results in terms of accuracy & AUC. The corresponding box-plot descriptive statistics is provided in Table VI and Table VII. The data shows that SMOTE provides the best median value of 82.4 & 0.765 for accuracy and AUC respectively.



(a) (b)

Fig. 3: Classifier Accuracy & AUC against different metrics-based datasets

Boxplot for four different feature selection techniques are illustrated in Figure 4. It is observed from Figure 4(a) and Figure 4(b) that SIG feature selection algorithm provides better results in terms of accuracy & AUC. The

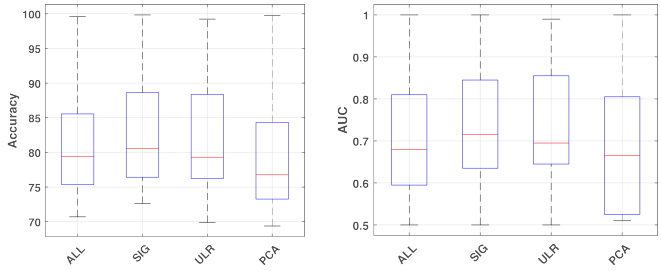
TABLE VI: Boxplot descriptive statistics : Classifier Accuracy against different datasets

	Min	Max	mean	median	std	var	Q1	Q3
OD	72.62	93.58	78.5005	76.99	5.332103039	28.43132282	75.12	79.895
SMOTE	69.38	99.83	84.22275	**82.4**	9.984943534	99.69909737	75.305	90.24

TABLE VII: Boxplot descriptive statistics : Classifier AUC against different datasets

	Min	Max	mean	median	std	var	Q1	Q3
OD	0.5	0.9	0.65525	0.655	0.102406618	0.010487115	0.59	0.715
SMOTE	0.5	1	0.772	**0.765**	0.161486683	0.026077949	0.64	0.89

corresponding box-plot descriptive statistics is provided in Table XI and Table XII. The data shows that SIG provides the best median value of 80.575 & 0.715 for accuracy and AUC respectively.



(a) (b)

Fig. 4: Classifier Accuracy & AUC against different datasets applying various feature selection techniques

Boxplot for ten different classification algorithms are illustrated in Figure 5. It si evident from Figure 5(a) and Figure 5(b) that LSSVM-PLY provides better results in terms of accuracy & AUC. The corresponding box-plot descriptive statistics is provided in Table XIII and Table XIV. The data shows that LSSVM-PLY provides the best median value of 96.405 & 0.945 for accuracy and AUC respectively.

TABLE VIII: p-value : Classification Algorithms

	LOGR	DT	ANNGD	ANNGDM	ANNGDMX	LSSVM-LIN	LSSVM-PLY	LSSVM-RBF	BTE	MVE
LOGR			•	•	•		•			
DT			•	•	•		•	•		
ANNGD				•		•	•	•	•	•
ANNGDM					•	•	•	•	•	•
ANNGDMX						•	•	•	•	•
LSSVM-LIN							•		•	
LSSVM-PLY								•		•
LSSVM-RBF										
BTE										
MVE										

In this study, ten different classification techniques are considered for analysis and a total of ${}^{10}C_2 = 45$ pairs are possible. Analysing all results at 0.05 significance level, we can reject a null hypothesis if and only if the p-value is less than $0.05/45 = 0.0011$. Upon observing the classification algorithms p-value in Table VIII, symbol " • " depicts the p-values less than 0.0011. There are 30 pairs of classification methods out of 45, that are significantly different (p-value < 0.0011) with better results. Out of these 30 pairs, it is observed that Least Squared Support Vector Machine with Polynomial kernel (LSSVM-PLY) yields superior results than other classification algorithms.

TABLE IX: p-value : Metrics-based dataset

	OD	SMOTE
OD		•
SMOTE		

Similarly, considering the p-value matrix for metrics-based dataset in Table IX. A total of ${}^{2}C_2 = 1$ pair is possible and analysing the result at 0.05 significance level, we can reject a null hypothesis if and only if the p-value is less than $0.05/1 = 0.05$. It is observed that the p-value less than 0.05 is depicted by symbol " • ". It is evident from Table IX, that class-balanced SMOTE dataset yields better results and is significantly different as compared to our original dataset (OD).

TABLE X: p-value : Feature selection algorithms

	ALL	SIG	ULR	PCA
ALL		•	•	•
SIG			•	•
ULR				•
PCA				

Considering four different feature selection algorithms as shown in Table X. A total of ${}^{4}C_2 = 6$ pairs are possible and analysing the result at 0.05 significance level, we can reject a null hypothesis if and only if the p-value is less than $0.05/6 = 0.0083$. It is observed that the p-value less than 0.0083 is depicted by symbol " • ". It can be inferred upon observing Table X that all of the feature selection techniques are significantly different and unique amongst themselves.

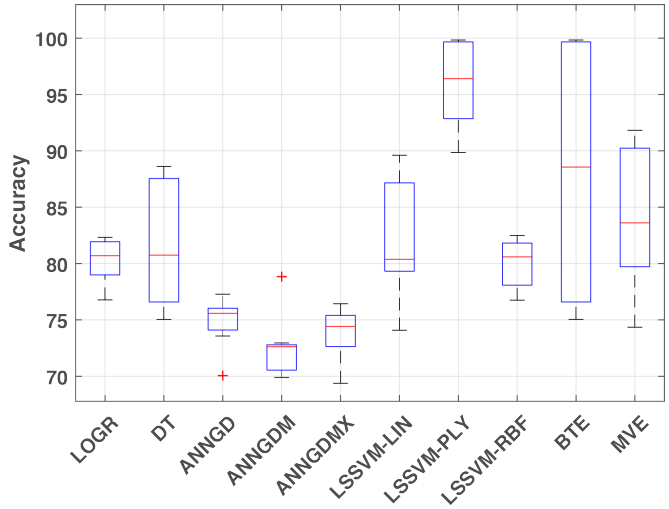
TABLE XI: Boxplot descriptive statistics : Classifier Accuracy against datasets applying various feature selection algorithms

	Min	Max	mean	median	std	var	Q1	Q3
ALL	70.7	99.61	81.4065	79.42	8.541362689	72.95487658	75.36	85.55
SIG	72.62	99.83	82.432	**80.575**	8.000261575	64.00418526	76.415	88.62
ULR	69.89	99.23	82.015	79.3	8.535067292	72.84737368	76.245	88.37
PCA	69.38	99.72	79.593	76.765	9.094804821	82.71547474	73.265	84.32

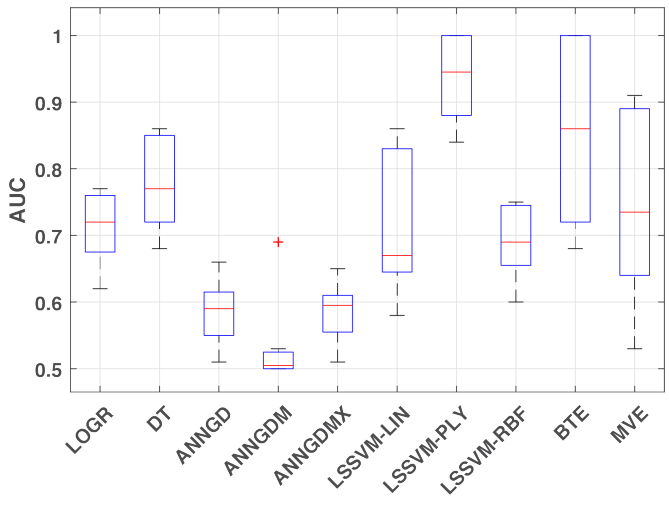
Finally, comparing our techniques with that of other authors as listed in Table XV, shows that our method

TABLE XII: Boxplot descriptive statistics : Classifier AUC against datasets applying various feature selection algorithms

	Min	Max	mean	median	std	var	Q1	Q3
ALL	0.5	1	0.7135	0.68	0.145756647	0.021245	0.595	0.81
SIG	0.5	1	0.7345	**0.715**	0.139037102	0.019331316	0.635	0.845
ULR	0.5	0.99	0.7285	0.695	0.145864934	0.021276579	0.645	0.855
PCA	0.51	1	0.678	0.665	0.159756393	0.025522105	0.525	0.805



(a)



(b)

Fig. 5: Classifier Accuracy & AUC box-plot

yields superior results with values of accuracy and AUC at 99.83% and 1.0 respectively. None of the authors in literature have used Least Squared Support Vector Machine with polynomial kernel function (LSSVM-PLY) over SMOTE-SIG Object-Oriented Metrics based dataset for android malware detection.

TABLE XIII: Boxplot descriptive statistics : Classifier Accuracy

	Min	Max	mean	median	std	var	Q1	Q3
LOGR	76.78	82.32	80.295	80.7	1.96974255	3.879885714	78.995	81.935
DT	75.04	88.62	81.67875	80.75	5.748675469	33.04726964	76.59	87.545
ANNGD	70.06	77.28	74.85	75.585	2.227868937	4.9634	74.105	76.04
ANNGDM	69.89	78.84	72.58125	72.62	2.799481776	7.837098214	70.55	72.79
ANNGDMX	69.38	76.43	73.84125	74.425	2.278642688	5.1922125	72.63	75.405
LSSVM-LIN	74.09	89.61	82.17625	80.375	5.375199498	28.89276964	79.32	87.16
LSSVM-PLY	89.86	99.83	95.94125	**96.405**	4.062470879	16.50366964	92.85	99.665
LSSVM-RBF	76.75	82.48	80.0275	80.59	2.215766556	4.909621429	78.085	81.82
BTE	75.04	99.83	88.06375	88.565	12.35587418	152.6676268	76.59	99.665
MVE	74.35	91.82	84.16125	83.605	6.394945409	40.89532679	79.715	90.24

TABLE XIV: Boxplot descriptive statistics : Classifier AUC

	Min	Max	mean	median	std	var	Q1	Q3
LOGR	0.62	0.77	0.7125	0.72	0.052847489	0.002792857	0.675	0.76
DT	0.68	0.86	0.7775	0.77	0.072456884	0.00525	0.72	0.85
ANNGD	0.51	0.66	0.585	0.59	0.05042675	0.002542857	0.55	0.615
ANNGDM	0.5	0.69	0.53125	0.505	0.065123509	0.004241071	0.5	0.525
ANNGDMX	0.51	0.65	0.585	0.595	0.045669621	0.002085714	0.555	0.61
LSSVM-LIN	0.58	0.86	0.71625	0.67	0.107827574	0.011626786	0.645	0.83
LSSVM-PLY	0.84	1	0.93625	**0.945**	0.067810134	0.004598214	0.88	1
LSSVM-RBF	0.6	0.75	0.69125	0.69	0.054099776	0.002926786	0.655	0.745
BTE	0.68	1	0.855	0.86	0.153063946	0.023428571	0.72	1
MVE	0.53	0.91	0.74625	0.735	0.146281285	0.021398214	0.64	0.89

TABLE XV: Comparison

Ref.	Accuracy	AUC
Proposed Work	**99.83%**	**1.0**
Zhuo Ma *et al.* [9]	-	-
Neeraj Chavan *et al.* [10]	97.0%	0.9900
Shivi Garg *et al.* [11]	98.27%	-
Yao-Saint Yen *et al.* [12]	92.0%	-
Alejandro Martín *et al.* [13]	77.8%	-
Dina Saif *et al.* [14]	99.1%	-
Ignacio Martín *et al.* [15]	92.7%	-
Xi Xiao *et al.* [16]	93.7%	-
Wei Wang *et al.* [17]	99.8%	-
Hui-Juan Zhu *et al.* [19]	89.91%	0.9031

VI. Conclusion

In this paper, we present techniques in order to classify android malware from benign apps. Initially, we consider five thousand seven hundred and seventy four android apps collected from various sources. These apps are decompiled using dex2jar and it's corresponding eighteen Object-Oriented metrics are obtained using CKJM extended tool. In order to tag an android app either as malware or benign, VirusTotal service from

Google Inc. is employed. Once the metrics-based dataset is obtained by clubbing together the Object-Oriented metrics and the malware tag, it is then verified whether considering Object-Oriented metrics as features will feasibly and effectively discriminate android malwares. To the best of our knowledge, we're the first ones to create Object-Oriented metrics based dataset. The effectiveness of Object-Oriented metrics to be used as potential features is validated using box-plot diagrams for each of the eighteen metrics. It is observed that there are less malware samples as compared to benign ones. Hence, SMOTE technique is applied over the original dataset in order to mitigate class-imbalance. Considering these two separate datasets *i.e.*, original dataset (OD) and class-balanced SMOTE dataset (SMOTE). Four different feature selection techniques such as ALL, SIG, ULR and PCA are applied over each of these datasets, consequently we obtain an additional of six different datasets *i.e.*, OD-SIG, SMOTE-SIG, OD-ULR, SMOTE-ULR, OD-PCA and SMOTE-PCA. Considering these six datasets along with OD and SMOTE, we have a total of eight different datasets. Subsequently, ten machine learning algorithms are used over these eight datasets to create $(10 \times 8 = 80)$ eighty models for malware classification. The discriminatory power of these classification models are then assessed using the classifier's accuracy and AUC. In this experiment, classifier's accuracy and AUC is described using box-plots. Various box-plots are depicted in order to capture three parameters such as dataset selection, feature selection and classifier selection affecting the overall malware detection accuracy and classifier AUC. Finally, it is observed from box-plot and its descriptive statistics that Least Squared Support Vector Machine (LSSVM-PLY) applied over SMOTE-SIG dataset yields better malware detection accuracy of 99.83% and AUC of 1.0.

In the future, additional Object-Oriented metrics will be included along with various machine learning algorithms such as deep learning. Currently, our experiment is a 2-class classification problem and only checks for benign and malware apps. Further, a multi-class classification model will be considered.

REFERENCES

[1] K. Nohl and K. Lell, "Mind the gap: Uncovering the android patch gap through binary-only patch level analysis," *HITB Security Conference*, 2018.

[2] P. O. Fora, "Beginners guide to reverse engineering android apps," in *RSA Conference*, 2014, pp. 21–22.

[3] S. R. Chidamber and C. F. Kemerer, "A metrics suite for object oriented design," *IEEE Transactions on software engineering*, vol. 20, no. 6, pp. 476–493, 1994.

[4] R. Martin, "Oo design quality metrics," *An analysis of dependencies*, vol. 12, pp. 151–170, 1994.

[5] J. Bansiya and C. G. Davis, "A hierarchical model for object-oriented design quality assessment," *IEEE Transactions on software engineering*, vol. 28, no. 1, pp. 4–17, 2002.

[6] B. Henderson-Sellers, *Object-oriented metrics: measures of complexity*. Prentice-Hall, Inc., 1995.

[7] M. H. Halstead *et al.*, *Elements of software science*. Elsevier New York, 1977, vol. 7.

[8] M.-H. Tang, M.-H. Kao, and M.-H. Chen, "An empirical study on object-oriented metrics," in *Proceedings sixth international software metrics symposium (Cat. No. PR00403)*. IEEE, 1999, pp. 242–249.

[9] Z. Ma, H. Ge, Y. Liu, M. Zhao, and J. Ma, "A combination method for android malware detection based on control flow graphs and machine learning algorithms," *IEEE Access*, vol. 7, pp. 21 235–21 245, 2019.

[10] N. Chavan, F. Di Troia, and M. Stamp, "A comparative analysis of android malware," *arXiv preprint arXiv:1904.00735*, 2019.

[11] S. Garg and N. Baliyan, "A novel parallel classifier scheme for vulnerability detection in android," *Computers & Electrical Engineering*, vol. 77, pp. 12–26, 2019.

[12] Y.-S. Yen and H.-M. Sun, "An android mutation malware detection based on deep learning using visualization of importance from codes," *Microelectronics Reliability*, vol. 93, pp. 109–114, 2019.

[13] A. Martín, V. Rodríguez-Fernández, and D. Camacho, "Candyman: Classifying android malware families by modelling dynamic traces with markov chains," *Engineering Applications of Artificial Intelligence*, vol. 74, pp. 121–133, 2018.

[14] D. Saif, S. El-Gokhy, and E. Sallam, "Deep belief networks-based framework for malware detection in android systems," *Alexandria engineering journal*, vol. 57, no. 4, pp. 4049–4057, 2018.

[15] I. Martín, J. A. Hernández, and S. de los Santos, "Machine-learning based analysis and classification of android malware signatures," *Future Generation Computer Systems*, 2019.

[16] X. Xiao, S. Zhang, F. Mercaldo, G. Hu, and A. K. Sangaiah, "Android malware detection based on system call sequences and lstm," *Multimedia Tools and Applications*, vol. 78, no. 4, pp. 3979–3999, 2019.

[17] W. Wang, M. Zhao, and J. Wang, "Effective android malware detection with a hybrid model based on deep autoencoder and convolutional neural network," *Journal of Ambient Intelligence and Humanized Computing*, pp. 1–9, 2018.

[18] A. Desnos and G. Gueguen, "Android: From reversing to decompilation," *Proc. of Black Hat Abu Dhabi*, pp. 77–101, 2011.

[19] H.-J. Zhu, T.-H. Jiang, B. Ma, Z.-H. You, W.-L. Shi, and L. Cheng, "Hemd: a highly efficient random forest-based malware detection framework for android," *Neural Computing and Applications*, vol. 30, no. 11, pp. 3353–3361, 2018.

[20] M. Jureczko and D. Spinellis, *Using Object-Oriented Design Metrics to Predict Software Defects*, ser. Monographs of System Dependability. Wroclaw, Poland: Oficyna Wydawnicza Politechniki Wroclawskiej, 2010, vol. Models and Methodology of System Dependability, pp. 69–81.

[21] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "Smote: synthetic minority over-sampling technique," *Journal of artificial intelligence research*, vol. 16, pp. 321–357, 2002.

[22] D. P. Mandic, A. I. Hanna, and D. I. Kim, "A general adaptive normalised nonlinear-gradient descent algorithm for nonlinear adaptive filters," in *2002 IEEE International Conference on Acoustics, Speech, and Signal Processing*, vol. 2. IEEE, 2002, pp. II–1353.

[23] S. Sahoo and M. K. Jha, "Pattern recognition in lithology classification: modeling using neural networks, self-organizing maps and genetic algorithms," *Hydrogeology journal*, vol. 25, no. 2, pp. 311–330, 2017.

[24] C.-C. Yu and B.-D. Liu, "A backpropagation algorithm with adaptive learning rate and momentum coefficient," in *Proceedings of the 2002 International Joint Conference on Neural Networks. IJCNN'02 (Cat. No. 02CH37290)*, vol. 2. IEEE, 2002, pp. 1218–1223.

[25] J. A. Suykens and J. Vandewalle, "Least squares support vector machine classifiers," *Neural processing letters*, vol. 9, no. 3, pp. 293–300, 1999.

[26] T. G. Dietterich, "An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization," *Machine learning*, vol. 40, no. 2, pp. 139–157, 2000.

[27] ——, "Ensemble methods in machine learning," in *International workshop on multiple classifier systems*. Springer, 2000, pp. 1–15.