Understanding Android Application Masquerades (Paper category: Preliminary)

Anand Tirkey, Ramesh Kumar Mohapatra
Computer Science and Engineering
National Institute of Technology
Rourkela, India
andy9c@gmail.com,rkmohapatra@ieee.org

Abstract

Android has revolutionized the smartphone industry since its unveiling in 2008. It has become the major choice of mobile operating system, modelled after Linux open-source ecosystem. This openness and general end-user accessibility has made android an open-ground for privacy breach and data theft, through masquerading rogue android apps. In this research paper we put forward a novel method to recognise masquerading android applications by deploying supervised machine learning algorithms over Object-Oriented software metrics based dataset. First, android apps are collected and decompiled into a repository. Object-Oriented software metrics are then calculated for each app using it's decompiled source codes, which forms the tuple of our dataset. Every tuple is tagged either as malware or benign using VirusTotal service. Consequently, this dataset is provided as input for machine learning algorithms. The malware recognising power for each machine-learned models can be evaluated using accuracy and AUC (area under ROC curve).

Keywords

Android, Malware Detection, Machine Learning, Object-Oriented Metrics

ACM Reference Format:
Anand Tirkey, Ramesh Kumar Mohapatra. 2020. Understanding Android Application Masquerades (Paper category: Preliminary). In *Innovations in Software Engineering Conference 2020, February 27–29, 2020, Indian Institute of Information Technology, Design and Manufacturing Jabalpur, India.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 Introduction

International Data Corporation (IDC, USA) forecasts android smartphone market share at around 87%, up from 85.1% in 2018, because of faster adoption of 5G enabled devices and expedited clearing of old stock inventory at competitive costs. In 2018, Symantec intercepted & blocked an average of 10,573 malicious mobile apps per day. In May 2019, Google reported that 53.5% of android devices run unsupported versions of the OS. Meanwhile, Karstern Nohl *et al*. [9] argue that, amongst the currently supported android versions, very few of the mobile device vendors truly provide timely bug fixes and critical security updates as published by Google. This has resulted in the android OS fragmentation as shown in Table 1, which exposes millions of end-users to serious malware threats.

Google has been working on "Project Treble" in order to reduce android os fragmentation. This project attempts to decouple vendor implementation (device-specific, lower-level software designed by silicon manufacturers) from the Android OS framework as shown in Figure 1 and Figure 2. This new architecture makes it easier, faster, and less costly for smartphone manufacturers to update devices to a new version of Android. Unfortunately, only those devices that ships with Android 8.0 and above are required to conform to these changes. Devices receiving Android 8.0 and above as upgrades, do not have to abide by these architectural improvements.Finally, end-users are at the discretion of the smartphone vendors for the software updates irrespective of how frequently Google releases critical security fixes for its supported Android OS.

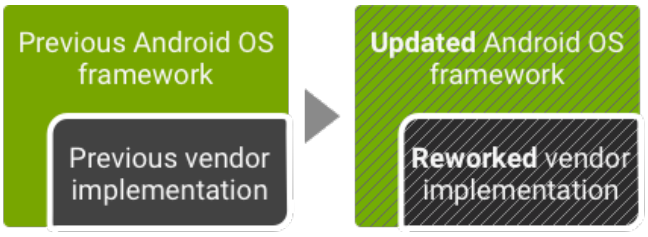


Figure 1: Legacy Android update environment

The rest of the paper is organized as follows. Section 2, discusses the proposed novel work, where android apps are collected and processed to extract its Object-Oriented Software metric, which forms the basis of our dataset. This dataset is used in training and testing various machine-learned models. Section 3, concludes the research findings along with looking forward into future directions, that could potentially improve android malware recognition.

ISEC '20, February 27–29, 2020, IIITDM Jabalpur, India

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

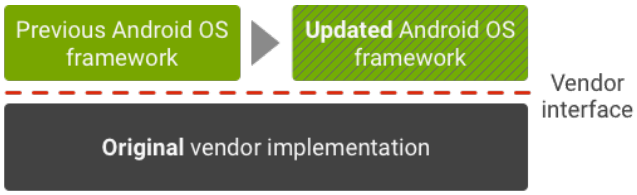


Figure 2: Current Android update environment

Table 1: 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%	
7.1		25	7.8%	Supported
8.0	Oreo	26	12.9%	
8.1		27	15.4%	
9.0	Pie	28	10.4%	

Table 2: 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

2 Proposed Work

The proposed work comprises of three primary components such as dataset collection phase, data pre-processing phase and android malware detection phase as illustrated in Figure 3.

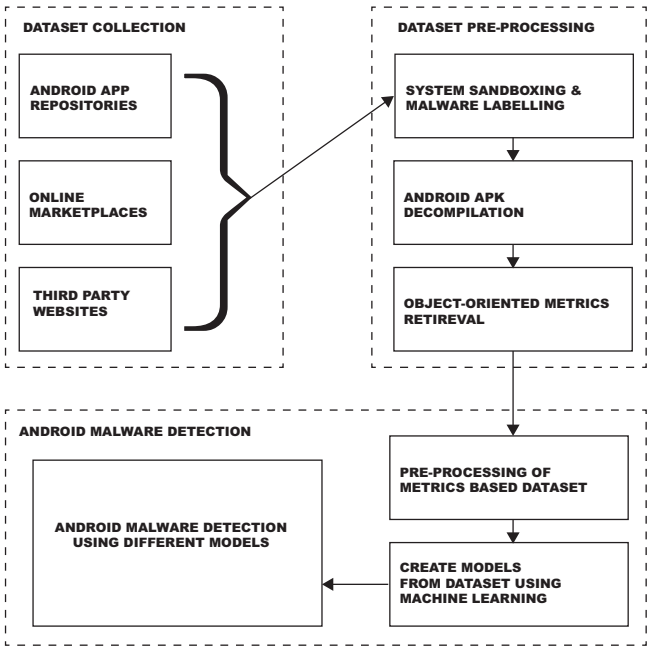


Figure 3: Object-Oriented Metrics based Android Malware Detection Model

2.1 Dataset Collection

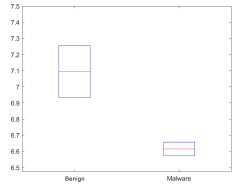
In this process, android application files are collected over internet from various sources like android repositories, official android marketplaces and third-party websites. The apk files are retrieved using both automated python scripts and manual downloading, which are then stored into an isolated repository for further processing. Finally, a total of five thousand seven hundred and seventy four different android apps are collected.

2.2 Dataset Pre-Processing

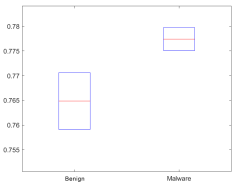
In this phase, a sandboxed system is used to work upon the collected android executable files. Initially, the apk files are decompiled to jar (java archive file) using tools such as dex2jar [4]. This decompiled jar artefact, is used to obtain the respective eighteen Object-Oriented Software Metrics as shown in Table 2, using CKJM-extended tool [7]. These Object-Oriented Software Metrics are defined by various authors such as Chidamber *et al*. [3], Martin *et al*. [8], Bansiya *et al*. [1], Henderson *et al*. [6], Halstead *et al*. [5] and Tang *et al*. [10]. The set of eighteen metrics form a tuple of our metrics-based dataset as illustrated in Table 3. This tuple is either marked as benign or malware using VirusTotal service from Google Inc. Finally, the set of these marked tuples, shape our metrics-based dataset, that will be used to train & test different machine-learned models.

Table 3: Metrics based Dataset Feature Vector

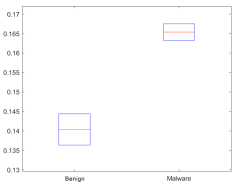
app_name	WMC	DIT	NOC	CBO	RFC	LCOM	Ca	Ce	NPM	...
...	LCOM3	LCO	DAM	MOA	MFA	CAM	IC	CBM	AMC	malware_tag



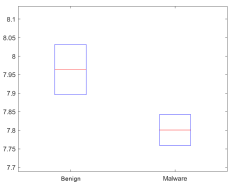
(a) WMC



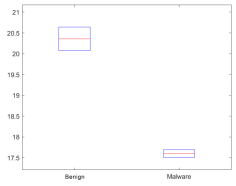
(b) DIT



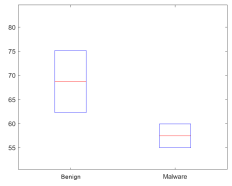
(c) NOC



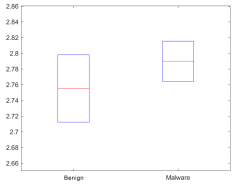
(d) CBO



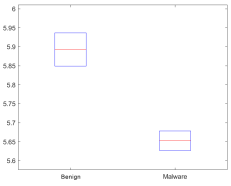
(e) RFC



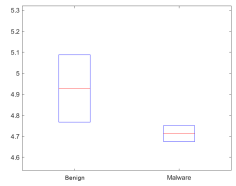
(f) LCOM



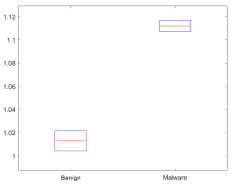
(g) Ca



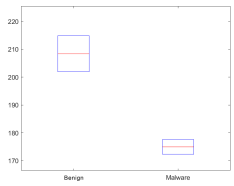
(h) Ce



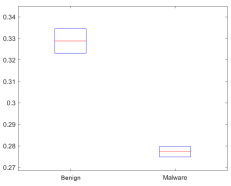
(i) NPM



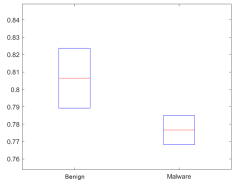
(j) LCOM3



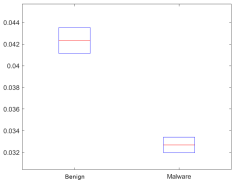
(k) LCO



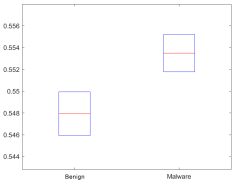
(l) DAM



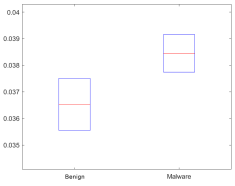
(m) MOA



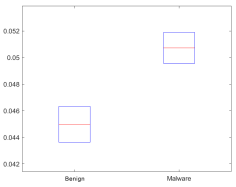
(n) MFA



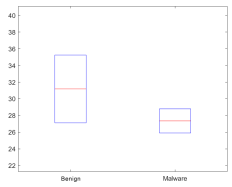
(o) CAM



(p) IC



(q) CBM



(r) AMC

Figure 4: Object-Oriented Metrics Boxplot

2.2.1 Effectiveness of Object-Oriented Metrics Before moving forward with the experiment, it is necessary to establish the feasibility and effectiveness in using this metrics-based dataset as input over various machine learning techniques, for improved android malware recognition. Boxplots for benign and malware apps, depicting each of the 18 Object-Oriented Metrics can be observed from Figure 4. Upon close observation at the inter-quartile ranges (IQR) from the boxplots, it is ascertained that the benign and malware IQR's for all of the Object-Oriented Metrics do not overlap except for Ca & AMC metrics as shown in Figure 4g and Figure 4r respectively. Therefore, the presence of multiple non-overlapping & distinct IQR patterns empower different machine learning algorithms, to discriminate malware from benign apps using Object-Oriented metrics based dataset.

$$IQR = (ThirdQuartile - FirstQuartile) \quad (1)$$

2.3 Android Malware Detection

Three different classification techniques are used such as logistic regression (LOGR), decision tree (DT) and artificial neural network with gradient descent-momentum (ANNGD) over five thousand seven hundred and seventy four different apk's, out of which one thousand five hundred and eighty two are malware apps. In order to mitigate this imbalance between malware and benign samples, SMOTE [2] (Synthetic Minority Oversampling Technique) has been employed, to generate class-balanced metrics dataset. The class-imbalanced metrics based dataset is termed as original dataset "OD" and the class-balanced metrics based dataset is termed as "SMOTE". We have selected features from both of these datasets using two ways such as ALL (considering all 18 Object-Oriented metrics as features) and SIG (wilcoxon signed-rank test) and as a result we have four different datasets as shown in Table 4. Finally, twelve different machine-learned models are formed using four datasets and three machine learning algorithms $(4 \times 3 = 12)$, where SMOTE-ALL dataset applied over decision tree (DT) yields a better accuracy and AUC of 88.95% and 0.87 respectively as shown in Table 5 and Table 6.

Table 4: Metrics-Based Datasets

1	OD-ALL	original dataset taking all features
2	SMOTE-ALL	class balanced smote dataset taking all features
3	OD-SIG	dataset obtained upon applying SIG over OD
4	SMOTE-SIG	dataset obtained upon applying SIG over SMOTE

3 Conclusion & Future Work

Initially, android apps are collected & decompiled thereby obtaining Object-Oriented Software Metrics from the decompiled artefacts, which forms the basis of our metrics-based dataset. The original class-imbalanced dataset "OD" and class-balanced dataset "SMOTE" are used over different machine learning techniques applying various feature selection techniques. Finally, SMOTE-SIG dataset applied over decision tree

Table 5: Classifier Accuracy against different datasets applying various feature selection techniques

	LOGR	DT	ANNGD
OD-ALL	79.41	77.95	76.31
SMOTE-ALL	82.67	**88.95**	74.34
OD-SIG	81.84	76.15	75.54
SMOTE-SIG	82.62	87.48	75.87

Table 6: Classifier AUC against different datasets applying various feature selection techniques

	LOGR	DT	ANNGD
OD-ALL	0.69	0.77	0.61
SMOTE-ALL	0.80	**0.87**	0.67
OD-SIG	0.75	0.71	0.59
SMOTE-SIG	0.73	0.86	0.61

yields better accuracy and AUC of 88.95% and 0.87 respectively. This research presents a binary classification problem, where all the different kinds of rogue apps are categorised under malware category and the rest are termed as benign. Further, classifying rogue apps into different malware families using Object-Oriented Metrics and android opcodes will be considered.

References

[1] Jagdish Bansiya and Carl G. Davis. 2002. A hierarchical model for object-oriented design quality assessment. *IEEE Transactions on software engineering* 28, 1 (2002), 4–17.

[2] Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. 2002. SMOTE: synthetic minority over-sampling technique. *Journal of artificial intelligence research* 16 (2002), 321–357.

[3] Shyam R Chidamber and Chris F Kemerer. 1994. A metrics suite for object oriented design. *IEEE Transactions on software engineering* 20, 6 (1994), 476–493.

[4] Pau Oliva Fora. 2014. Beginners guide to reverse engineering android apps. In *RSA Conference*. 21–22.

[5] Maurice Howard Halstead et al. 1977. *Elements of software science*. Vol. 7. Elsevier New York.

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

[7] Marian Jureczko and Diomidis Spinellis. 2010. *Using Object-Oriented Design Metrics to Predict Software Defects*. Monographs of System Dependability, Vol. Models and Methodology of System Dependability. Oficyna Wydawnicza Politechniki Wroclawskiej, Wroclaw, Poland, 69–81.

[8] Robert Martin. 1994. OO design quality metrics. *An analysis of dependencies* 12 (1994), 151–170.

[9] Karstern Nohl and Kajob Lell. 2018. Mind the Gap: Uncovering the Android Patch Gap Through Binary-Only Patch Level Analysis. *HITB Security Conference* (2018).

[10] Mei-Huei Tang, Ming-Hung Kao, and Mei-Hwa Chen. 1999. An empirical study on object-oriented metrics. In *Proceedings sixth international software metrics symposium (Cat. No. PR00403)*. IEEE, 242–249.