Dynamic Permissions based Android Malware Detection using Machine Learning Techniques

Arvind Mahindru*
Department of Computer Science & Applications
D.A.V. University
Sarmastpur 144012, Jalandhar
er.arvindmahindru@gmail.com

Paramvir Singh
Department of Computer Science & Engineering
Dr. B.R. Ambedkar National Institute of Technology, Jalandhar 144001
singhpv@nitj.ac.in

ABSTRACT

Android is by far the most widely used mobile phone operating system around. However, Android based applications are highly vulnerable to various types of malware attacks attributed to their open nature and high popularity in the market. The fault lies in the underneath permission model of Android applications. These applications need a number of sensitive permissions during their installation and runtime, which enables possible security breaches by malware. The contributions of this paper are twofold: 1) We extract a set of 123 dynamic permissions from 11000 Android applications in a largest publicly available dataset till date; 2) We evaluate a number of machine learning classification techniques including Naive Bayes (NB), Decision Tree (J48), Random Forest (RF), Simple Logistic (SL), and k-star on the newly designed dataset for detecting malicious Android applications. The experimental results indicate that although the malware classification accuracy of RF, J48, and SL are comparable, SL performs marginally better than the other techniques.

Keywords

Android; Malware Detection; Machine Learning; Dynamic Analysis

1. INTRODUCTION

Mobile phones with advance computing capabilities and better connectivity than regular mobile phones, came into the market in late 90's, but gained popularity with the introduction of Android operating system. Android is currently the most popular smartphone platform which occupied 86.2% of global sale by the end of first half of 2016 [26]. Google launched 20 versions of Android operating system from the year 2008 to year 2016 in the market. However, the most popular version of Android is 4.4 (kitkat), which covers globally the share of 32.5% of smartphones running in the market [28]. By the end of June 2016, 2.2 million applications are available in Android market [29] and more than one million of these applications are used by smartphone users. With the popularity of Android applications, it can also invite cyber criminals to develop malicious applications that can steal information from the smartphones. According to a study in [27], G DATA security expert analyzed 1,723,265 new malware samples in the first half of 2016. This is an increase of 29% as compared to the second half of 2015. It means that in every nine seconds, a new malicious application is being introduced in the Android market. These applications are created to launch different types of attacks in the form of trojans, backdoor, worms, botnet and spyware. The data presented in [16] shows that there is an average increase from 39 to 50 malware variants per family from April 2015 to March 2016.

Android systems have a permissions based mechanism to enforce security restrictions on applications. The installation and start-up of an Android application requires a number of permissions like read contacts, internet access, enable or disable application components etc. The installer package shows the lists of permissions where a user can allow or deny these permissions. When a user sets the permission then it can easily access the resource and it is not possible to revoke these permissions until the application is uninstalled. In recent times, researchers have proposed three different approaches for android malware detection: static, dynamic and hybrid [6].

Static analysis methods examine the code without actually executing it, hence they are quick but have to deal with many false-positives. This technique has a major disadvantage of code obfuscation and dynamic code loading. On the other hand, dynamic analysis techniques monitor the implemented code and inspect its interaction with the system. The main advantage of this technique is that it detects dynamic code loading and records the application behavior during run time. Though they are time-consuming, but they are effective against malware obfuscation. So, in this paper, we use dynamic analysis approach to detect malware in Android applications. We perform an analysis on a set of 11,000 Android application packages (*.apk*) to collect the permissions required by these applications at the time of installation and start-up. To built our dataset, we divide these applications into their respective domains (such as arcade and action, comics, entertainment etc.) and further classify them as normal and malware. The major contributions of this work are as follows:

*Corresponding Author

ISEC '17, February 05-07, 2017, Jaipur, India

DOI: http://dx.doi.org/10.1145/3021460.3021485

Table 1: **Brief description of some earlier derived techniques with their accuracy**

Technique	Goal	Description	Methodology	Data set	Accuracy
Drebin [1]	Detection	Gathering as many features of an application as possible	Static	123,453N 5560M	94%
Androsimilar [8]	Detection	Extracting Statistically improbable features	Static	7324N 1260M	99.4% for Google Play apps & 99.89% for 3rd party apps
PUMA [14]	Detection	Permission Usage to detect Malware in Android	Dynamic	239M	86.30%
Andromaly [15]	Detection	Host based malware detection system	Dynamic	4M	100% for self-written malware
DroidMat [19]	Detection	Android Malware Detection through API calls	Static	1500N 238M	97.87%

Here "N" means normal permissions and "M" means malware permissions

- A dataset comprising installation and start-up (dynamic) time permissions extracted from 11,000 normal and malware Android application packages (*.apk*) collected from multiple sources.
- Evaluation of the constructed dataset using a set of five machine learning classification techniques.

Rest of the paper is summarized as follows. Section 2 describes the related work. In Section 3 we describe our experimental methodology. Section 4 contains the evaluation of data set. Section 5 contains the discussion on the comparison of our methodology with the existing methodology. Section 6 distinguishes threats to validity. Section 7 describes the future scope and conclusion.

2. RELATED WORK

There are a number of approaches proposed for malware detection based on static, dynamic and hybrid analysis. Chin et al. [4] proposed Comdroid for detecting application communication based vulnerabilities in Android. Grace et al. [12] proposed a proactive scheme to spot zero-day Android malware. Fuchs et al. [9] proposed ScanDroid technique which analyzes the data policies in an application manifest and data flows across content providers. Barrera et al. [2] proposed a methodology for identifying application clusters based on requested permissions at their installation time. Zhou et al. [24] proposed a permission based behavioral approach to detect the malware.

Zhao et al. [22] suggested AntiMalDroid to detect android malware, that uses behaviour sequence as the feature. Enck et al. [10] proposed TaintDroid technique, which is used to detect the features like tracking of variables, methods and files used by the malware. Burguera et al. [3] presented a clustering technique named Crowdroid to detect the malware. Dini et al. [5] proposed MADAM, which uses the dynamic approach with detection at both kernel and user levels. Wu et al. [20] proposed, DroidDolphin approach to create log files and extract information from it to protect them from malware. A VM based dynamic system call-centric analysis and stimulation technique called CopperDroid is proposed in [17]. Shabtai et al. [15] proposed Andromaly that recognize a host based malware detection system that continuously monitors various features and events obtained from the mobile device and then applies anomaly

Table 2: Categories of Android application packages (*.apk*)

Category	N	T	B	W	BO	SP
Arcade and Action	280	220	50	102	60	58
Books and Reference	58	90	78	85	156	72
Brain and Puzzle	192	82	154	10	125	120
Business	176	152	89	56	14	35
Cards and Casino	299	76	65	81	35	42
Casual	325	321	69	46	50	50
Comics	295	65	95	35	18	0
Communication	325	52	50	50	50	50
Education	265	56	89	65	0	35
Entertainment	198	0	225	120	21	60
Finance	125	5	200	99	65	36
Health and Fitness	325	98	65	45	0	0
Libraries and Demo	89	98	65	89	65	103
Lifestyle	100	155	200	100	0	0
Media and Video	100	100	123	162	0	58
Medical	100	123	135	125	0	25
Music and Audio	322	65	0	65	0	0
News and Magazines	37	0	0	0	0	0
Personalization	400	0	42	25	25	0
Photography	18	5	12	0	0	0
Productivity	0	100	16	0	0	0
Racing	0	50	10	21	0	0
Shopping	0	0	0	20	13	0
Social	0	0	50	20	0	0
Sports	0	0	24	0	0	0
Sports Games	0	0	45	45	0	0
Tools	0	120	30	5	5	0
Transportation	0	2	2	0	1	0
Travel and Local	0	0	22	0	41	1
Weather	0	12	23	0	0	0

Here, "N" means Normal, "T" means Trojan, "B" means Backdoor, "W" means Worms, "BO" means Botnet, and "SP" means Spyware

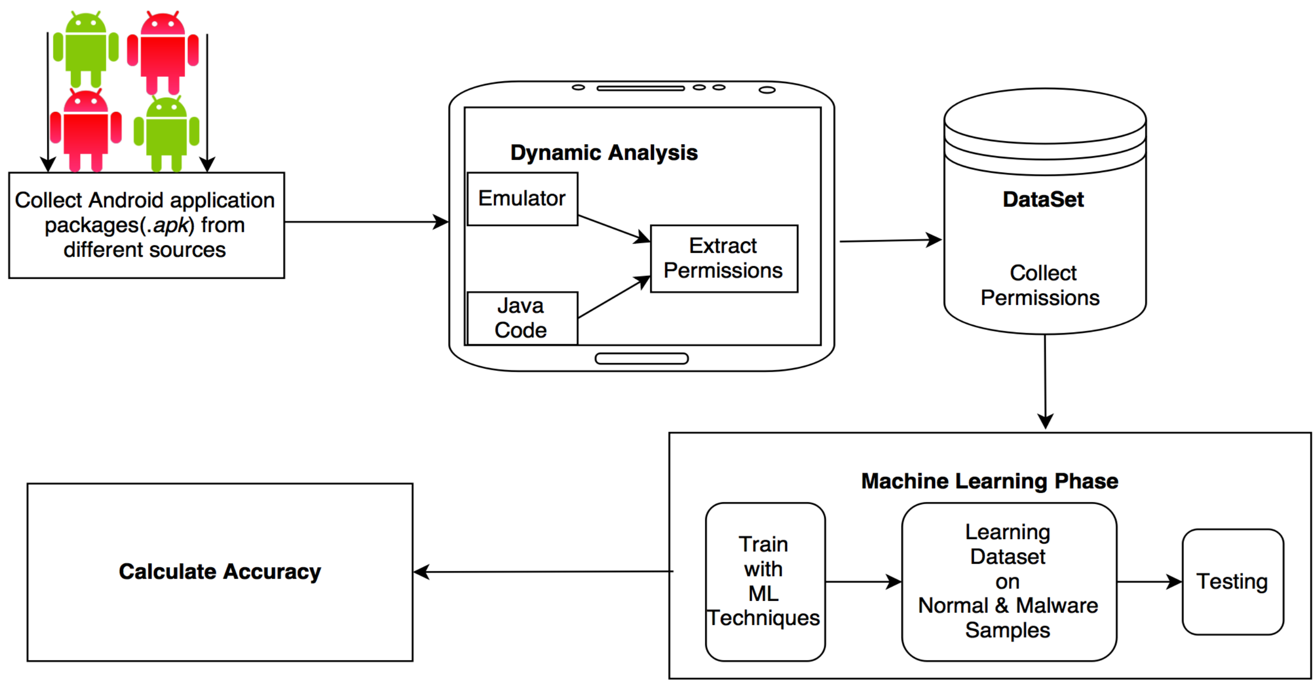

Figure 1: Proposed Methodology

detectors to classify the collected data as normal or abnormal. Faruki et al. [8] proposed AndroSimilar, a robust approach which generates signature information by extracting statistically improbable features to detect malicious Android applications. Xu et al. [21] proposed Aurasium which automatically repackages arbitrary applications to attach user-level sandboxing and policy enforcement code, which closely watches the applications behavior for security and privacy violations. Google play manages its security using a system called Bouncer [6, 18], which is a virtual machine based dynamic analysis platform to test the uploaded third party developer applications before availing them to the users for download. It executes applications to look for any malicious behaviour, and also compares it against previously analysed malware applications.

Zhou et al. [25] proposed Droidranger, which uses both the static and dynamic approaches to detect malware. Martina et al. [13] proposed the technique Andrubis which uses both static and dynamic analysis for network operations, data leaks, SMS and phone calls to protect for data leaks.

Table 1 lists the underneath methodology, dataset size, and accuracy of some recent works on the detection of Android malwares. These works either follow a static approach or dynamic approach, both providing good degree of accuracy. However, due to the experimentation over small size dataset, the accuracy value attained by these works is not applicable to a wide range of scenarios. In this work, we construct a dataset of the most recently launched 11,000 normal and malware applications, and apply various machine learning algorithms to evaluate its effectiveness in identifying malware applications.

3. EXPERIMENTAL METHODOLOGY

The experimental methodology presented in this paper is divided into three phases as shown in Figure 1. In the first phase, the android application packages (*.apk*) are collected from different sources. In the second phase, dynamic analysis is performed on these collected Android application packages (*.apk*) and the permissions which are used by these applications during their installations and start-up are collected to form our data set. In the last phase, we evaluate our data set by implementing various machine learning algorithms.

3.1 Collection of (*.apk*)

In the first phase of our methodology, we collect 13,000 unique Android application packages (*.apk*) comprising of 6029 normal Android application packages (*.apk*) from apchina [34], hiapk [35], android [36], mumayi [37], gfan [38], pandaapp [39], slideme [40] and 6971 malicious applications from different sources such as Android Botnet data set [7], DroidKin data set [11], Android Malware Genome Project [23] and AndroMalShare [33]. The considered normal applications were launched in Feb. 2014 to Feb. 2016 and the malware applications were introduced in between March 2014 to March 2016. The malicious applications come from different malware families such as trojan, backdoor, worm, botnet and spyware. The collected Android application packages(*.apk*) are categorized in 29 different categories as shown in Table 2.

3.2 Extract Permissions from collected (*.apk*)

We run these collected Android application packages (*.apk*) with the help of emulator bluestack [41]. Further, we extract permissions by running a java code and made our data set. Example of extracted permissions are given below:

```
uses-permission: android.permission.READ_CALENDER_EVENT
uses-permission: android.permission.WRITE_CONTACTS
```

By applying dynamic analysis on collected 13,000 Android application packages (*.apk*), we discard the ones that were

Table 3: Safe permissions with apps count

Safe permissions with their field	# apps
D: access DRM content	8
D: access email provider data	9
D: access all system downloads	3
D: access download manager	7
D: advanced download manager functions	1
D: install DRM content	14
D: modify Google service configuration	2
D: modify Google settings	2
D: move application resources	1
D: read Google settings	3
D: send download notifications	5
D: voice search shortcuts	2
D: access surface flinger	7
D: access checkin properties	7
D: access the cache file system	1
D: act as an account authenticator	5
D: bind to a wallpaper	17
D: bind to an input method	2
D: change screen orientation	32
D: control system backup and restore	3
D: delete applications	11
D: delete other applications caches	12
D: delete other applications data	13
D: directly call any phone numbers	56
D: directly install applications	43
D: disable or modify status bar	20
D: display unauthorized windows	5
D: enable or disable application components	11
D: force application to close	2
D: force device reboot	12
D: full Internet access	68
D: interact with a device admin	4
D: manage application tokens	1
D: modify battery statistics	65
D: modify secure system settings	57
D: modify the Google services map	1
D: monitor and control all application launching	35
D: partial shutdown	1
D: permanently disable device	5
D: power device on or off	5
D: press keys and control buttons	6
D: prevent app switches	5
D: read frame buffer	5
D: read instant messages	5
D: record what you type and actions you take	2
D: reset system to factory defaults	5
D: run in factory test mode	5
D: set time	5
D: set wallpaper size hints	1
D: start IM service	5
D: update component usage statistics	1
D: write instant messages	5
DT: enable application debugging	1
DT: limit number of running processes	5
DT: make all background applications close	5
DT: send Linux signals to applications	2
HC: change your audio settings	336
HC: control flashlight	165
HC: control vibrator	2555
HC: take pictures and videos	547
HC: test hardware	17
NC: broadcast data messages to applications	5
NC: control near field communication	12
NC: create bluetooth connections	220
NC: download files without notification	5
NC: receive data from Internet	389

Table 4: Dangerous permissions with apps count

Dangerous permissions with their field	# apps
D: Audio File Access	3
D: access to passwords for Google accounts	5
D: coarse (network-based) location	5
D: control location update notifications	18
D: discover known accounts	5
D: mock location sources for testing	10
D: modify/delete USB storage contents modify/delete SD card contents	10
D: permission to install a location provider	4
D: read phone state and identity	1
D: write contact data	10
HC: record audio	471
NC: full Internet access	9017
NC: make/receive Internet calls	3
NC: view Wi-Fi state	1438
NC: view network state	6880
PC: intercept outgoing calls	100
PC: modify phone state	100
PC: read phone state and identity	4368
SCM: directly call phone numbers	625
SCM: send SMS messages	337
S: modify/delete USB storage contents modify/delete SD card contents	4337
ST: change Wi-Fi state	235
ST: format external storage	4
YA: Blogger	50
YA: Google app engine	100
YA: Google docs	100
YA: Google finance	100
YA: Google maps	100
YA: Google spreadsheets	100
YA: Google voice	100
YA: Google mail	2
YA: Picasa web albums	100
YA: YouTube	100
YA: YouTube usernames	100
YA: access all Google services	100
YA: access other Google services	100
YA: act as an account authenticator	7
YA: act as the Account manager service	4
YA: contacts data in Google accounts	10
YA: discover known accounts	27
YA: manage the accounts list	11
YA: read Google service configuration	10
YA: use the authentication credentials of an account	2
YA: view configured accounts	1
YL: access extra location provider commands	5
YL: coarse (network-based) location	20
YL: fine (GPS) location	23
YL: mock location sources for testing	3
YM : edit SMS or MMS	8
YM : read SMS or MMS	10
YM: receive SMS	15
YM: receive WAP	1
YM: send SMS-received broadcast	1
YPI: add or modify calendar events and send email to guests	7
YPI: read calendar events	1
YPI: read contact data	24
HC: write contact data	18

Table 5: Machine Learning Classifiers used in the experiment.

Algorithms	Configuration
Naive Bayes (NB)	N/A
Decision Tree (J48)	Size of tree=19
Random Forest (RF)	8 random features
Simple Logistic	Number of classes =2
k-star	k=1

either not installed or do not start from their launch menu. From rest of the Android application packages(*.apk*) we collect 123 unique permissions which these applications require. By collecting these permissions, we construct our data set of 11,000 unique applications. Before moving to the implementation phase, we divide these 123 permissions into safe and unsafe states. These safe and unsafe permissions [30, 31, 32] are mentioned in Tables 3 and 4 respectively. In Table 3 and 4 we divide the field in which Android applications require permissions at installation time and start-up. In these tables, "D" stands for Default permissions, "DT" stands for Development tools, "HC" stands for Hardware control, "NC" stands for Network communications, "PC" stands for Phone calls, "SCM" Services that costs money, "S" stands for storage, "ST" stands for System tools, "YA" stands for Your accounts, "YM" stands for Your messages and "YPI" stands for Your personal information. Default permissions are the ones which the Android applications require at their installation time, Development tools, Hardware control, Network communications, Phone calls, Services that costs money and Your messages are the permissions which Android applications require at their start-up and Storage, System tools, Your accounts and Your personal information are the permissions that are required by the Android applications at their installation time and start-up.

3.3 Machine Learning Phase

We evaluate our data set by using five machine learning classifier techniques which are: Naive Bayes (NB), Decision Tree (J48), Random Forest (RF), Simple Logistic and k-star. The five classifiers with their configuration which is used in our experiments are shown in Table 5.

Naive Bayes: This technique uses the method of conditional independence assumption in which training phase require considering each attribute in each class separately and in testing phase it calculate conditional probability with estimate distribution.

Decision Tree: This uses the method of discrete value target function, in which the learned function is represented as a set of if-then rules. Decision Tree consists of nodes that form a rooted tree, with a node called root. A node with outgoing is called a test node. In our experiment we consider the size of trees 19.

Random Forest: RF consists of a collection of tree structured classifier $\{h(X, \Theta_k), k = 1, 2, 3, \ldots\}$, where the $\{\Theta_k\}$ are independent identically distributed random vector and each tree cast a vote for the most popular class at input X. In our experiment we consider 8 random features.

Simple Logistic: It can interpret prediction of class membership. By applying class assignment threshold probability. In our experiment we consider number of classes 2.

Table 6: Testing of training and supply data set with all classifiers

Techniques	TPR	FPR	Prec.	Recall	F-Measure
Naive Bayes	0.987	0.007	0.988	0.987	0.987
J48	0.996	0.003	0.996	0.996	0.996
Random Forest	0.996	0.002	0.996	0.996	0.996
Simple Logistic	0.997	0.002	0.997	0.997	0.997
k-star	0.952	0.028	0.957	0.952	0.952

Table 7: Testing during cross validation of data set with all classifiers

Techniques	TPR	FPR	Prec.	Recall	F-Measure
Naive Bayes	0.984	0.009	0.985	0.984	0.984
J48	0.996	0.003	0.996	0.996	0.996
Random Forest	0.996	0.003	0.996	0.996	0.996
Simple Logistic	0.996	0.003	0.997	0.996	0.996
k-star	0.952	0.028	0.957	0.952	0.952

k-star: k-star is a lazy learning method in which generalization beyond the training data is delayed until a query is made to the system, as opposed to an eager learning, where the system tries to generalize the training data before receiving queries. In our experiment we consider k=1.

4. EVALUATION OF DATA SET

In order to achieve the accuracy as high as possible, we test our data set in WEKA [42] by using three different options available in it. First option is supply training set and then evaluate the test set, the second is cross-validation and third is splitting the data set on the percentage basis. In the following subsection, we define the parameters True Positive Rate, False Positive Rate, Precision, F-measure and accuracy. To show the effectiveness of our approach, we shall evaluate each parameter for all the five considered classification algorithms, and for each mode of testing the data set.

4.1 Evaluation Measures

To assess the effectiveness of the selected classification algorithms, we calculate the metrics true positive, false positive, recall, accuracy, precision rate and F-measure in our experiments. To define these parameters firstly let us consider that TP (true positive) be the number of Android malware that are correctly detected; FN (false negative) be the number of Android malware that are not detected (predicted as benign application); let TN (true negative) be the number of benign applications that are correctly classified; and FP (false positive) be the number of benign applications that are incorrectly detected as Android malware.

Table 8: Testing of splitting data set with all classifiers

Techniques	TPR	FPR	Prec.	Recall	F-Measure
Naive Bayes	0.987	0.007	0.988	0.987	0.987
J48	0.997	0.002	0.997	0.997	0.997
Random Forest	0.997	0.002	0.997	0.997	0.997
Simple Logistic	0.997	0.002	0.997	0.997	0.997
k-star	0.952	0.028	0.958	0.952	0.953

True Positive Rate: It measures the proportion of positives that are correctly identified and is given by

$$TruePositiveRate(TPR) = \frac{TP}{TP + FN}$$

False Positive Rate: It measures the proportion of negatives that are correctly identified and is defined as

$$FalsePositiveRate(FPR) = \frac{FP}{TN + FP}$$

Precision: The proportion of the actual malicious apps are correctly classified to the total of all apps that are classified as malicious.

$$Precision(Prec) = \frac{TP}{TP + FP}$$

Recall Rate: The proportion of the malicious apps that are classified correctly to the total number of the malicious that are classified correctly as malicious or incorrectly as benign.

$$RecallRate = \frac{TP}{TP + FN}$$

F-measure: The harmonic mean of precision and recall. This value tells how much the model is discriminative.

$$F - measure = \frac{2 \times Recall \times Precision}{Recall + Precision}$$

Accuracy: The proportion of the total number of the apps that are correctly classified whether as benign or malicious.

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN}$$

Now, in the following sections we consider the three modes of testing the data set and calculate the evaluation parameters for each classifiers.

4.2 Training and Supply data set

Training set means that file is loaded first is preprocess for testing and then we supplied a data set for evaluation. 70% of the data set was used for training the classifier, and 30% was used for testing. The five metrics TPR, FPR, $Prec.$, $Recall$ and $F - measure$ calculated for this mode of supplying the data set are shown in Table 6. The percentage accuracy for all the classifiers is presented in Figure 2, which shows that Simple Logistic attains the highest accuracy. The classifiers report the accuracy rank-wise as following: 0.997, 0.996, 0.996, 0.987 and 0.952 for Simple Logistic, J48, Random Forest, Naive Bayes and k-star respectively. Further, we also see the Android malware detection analysis comparison in the considered algorithms from the Precision-Recall view. This is shown in Figure 3.

4.3 Cross Validation

We perform 10-fold cross-validation in which WEKA divide the data set into 10 parts (these are called "folds"), hold out each part in turn, and then average the results. So each data point in the data set is used once for testing and 9 times for training. Table 7 shows the results which are obtained by using five different classification techniques of machine learning. Figure 4 represents the percentage accuracy obtained by all the five classifiers. As can be seen from Figure 4 that the three classifiers J48, Random Forest and Simple Logistic reports the equal and highest accuracy of

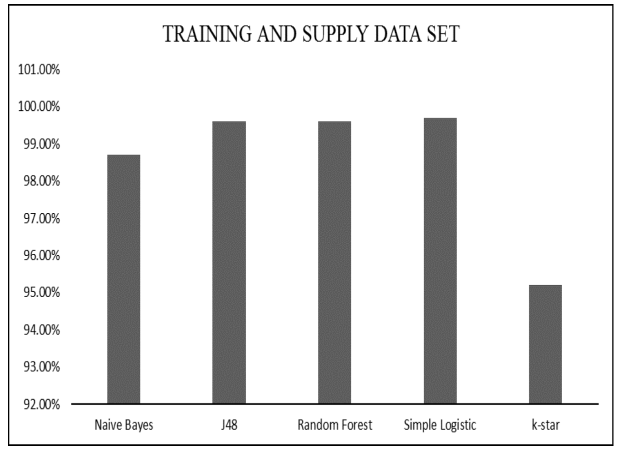

Figure 2: Accuracy of training and supply dataset for all classifiers

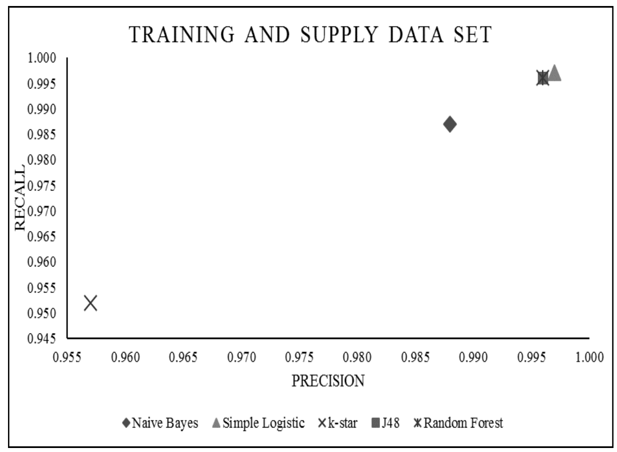

Figure 3: Android malware detection analysis comparison for training and supply dataset in all classifiers from the Precision-Recall view.

99.6%. Naive Bayes obtained the second highest accuracy of 98.4% and an accuracy of 95.2% is reported by k-star. The Precision-Recall view to show the comparison between the classifiers for Android malware detection is shown in Figure 5. The point (Precision, Recall)=(0.996, 0.996) for J48 and Random Forest is almost same as the point (Precision, Recall)=(0.997, 0.996) for Simple Logistic, so the points in the graph are overlapped.

4.4 Splitting Data set

The third method is to split our data set on percentage basis, which means that classification results is evaluated on a test set that is a part of the original data. For evaluation we split our data set by 66%. The five metrics $TPR, FPR, Prec., Recall$ and $F - measure$ calculated for this mode of supplying the data set are shown in Table 8. In Figure 6, the accuracy for splitting data set is given. The Figure 6 shows that like the previous case J48, Random Forest, Simple Logistic attains the equal and highest accuracy of 99.6%. The next highest accuracy of 98.7% is obtained by Naive Bayes and lastly 95.2% accuracy by k-star is achieved. Figure 7, represents the Precision-Recall view for compari-

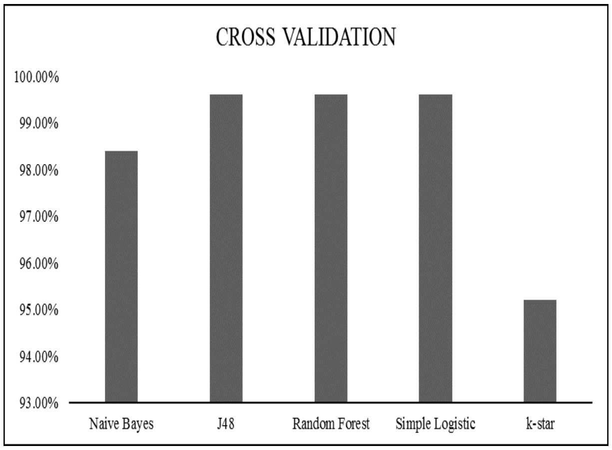

Figure 4: Accuracy of cross validation for all classifiers

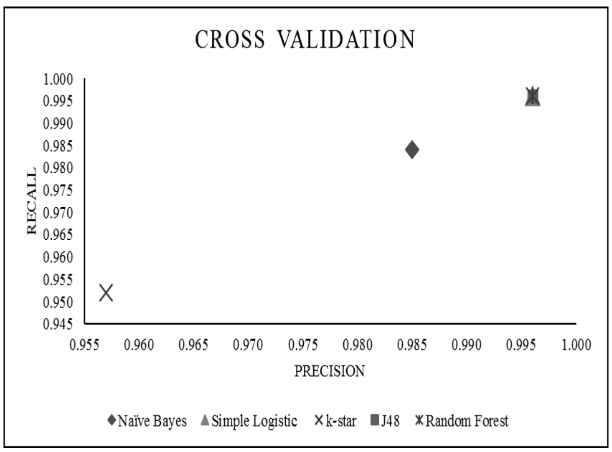

Figure 5: Android malware detection analysis comparison for cross validation in all classifiers from the Precision-Recall view.

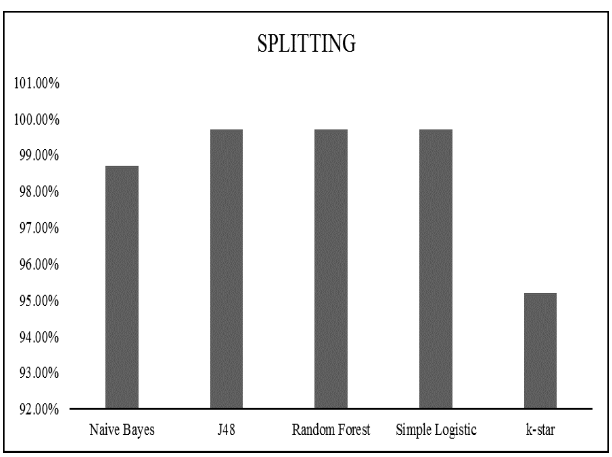

Figure 6: Accuracy of splitting data set for all classifiers

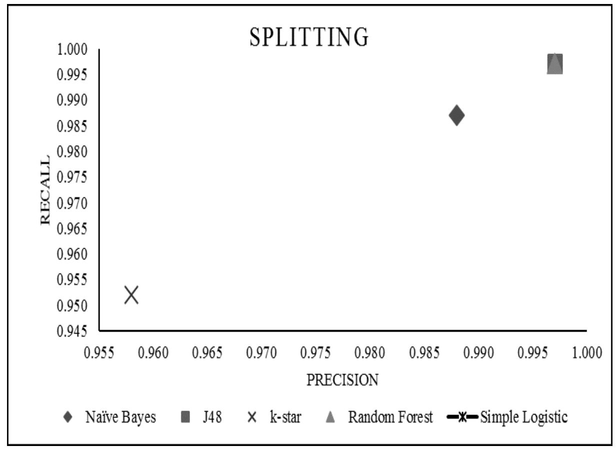

Figure 7: Android malware detection analysis comparison for splitting in all classifiers from the Precision-Recall view.

son of the classifiers.

5. DISCUSSION

We evaluate our dataset consisting of 11,000 (out of which 6,971 are malware samples) Android applications by applying five machine learning classifier techniques. By applying, it on our dataset we achieve highest accuracy rate of 99.7% to detect malware by simple logistic technique. Android malware detection technique PUMA [14] which uses only 239 malware applications in the dataset achieve an accuracy of 84.08% with Simple Logistic, 86.41% with Random Forest 50, 67.64% with Naive Baiyes and 81.32% with J48. These accuracy rates for each technique are much lesser than the achieved for our dataset as shown in figures 2, 4 and 6.

6. THREATS TO VALIDITY

Applications not require permission: While constructing data set for evaluation, some of the Android application packages (*.apk*) do not require any kind of permission at installation and start-up. So during implementation in WEKA almost 1% of applications were included from the total number of applications considered.

Application Crashes: At the time of dynamic analysis, some applications were installed in our emulator but they does not start at start-up time. So, approximately 2% of such applications were included in our dataset.

Permission overcome: Some of the normal and malware application require the same permissions and these applications are included in our dataset.

7. CONCLUSION AND FUTURE SCOPE

In this paper, we evaluate 11,000 Android applications by applying various machine learning techniques like Naïve Bayes, J48, Random Forest, Simple Logistic and k-star and it is seen that the highest accuracy rate of 99.7% is achieved using the Simple Logistic machine learning technique while using the option training and supply data set. During evaluation in cross validation and splitting data set option J48, Random Forest and Simple Logistic machine learning technique attain the highest accuracy. In nut shell, we can say that Simple Logistic technique is capable to discriminate almost all the cases of malware existing in the considered data set.

Furthermore, day-by-day new applications are adding in Android market. So we can enhance the dataset. Also, we can implement various other machine learning techniques on our dataset. This data set can also be applied on cloud based or clustering malware detection system.

Availability: Our dataset, is publicly available at http://pvsingh.com/a_mahindru. for use of researchers.

8. REFERENCES

[1] H.A. Alatwi. Android malware detection using category-based machine learning classifiers. *Masters thesis*, June 2016.

[2] D. Barrera, H.G. Kayacik, P.C. van Oorschot and A. Somayaji. A methodology for empirical analysis of permission-based security models and its application to android. *In Proc. of conference on Computer and communication security,* ACM, October 2010.

[3] I. Burguera, U. Zurutuza, and S. Nadjm- Tehrani. Crowdroid: Behavior-based malware detection system for android. *In Proc. of the 1st Workshop on Security and Privacy in Smartphones and Mobile Devices,* CCSSPSM'11, 2011.

[4] E. Chin, A.P. Felt, K. Greenwood, and D. Wager. Analyzing inter-application communication in android. *In proc. of International conference on Mobile systems, applications and services,* ACM, 2011.

[5] G. Dini, F. Martinelli, A. Saracino, and D. Sgandurra. MADAM: A multi-level anomaly detector for android malware. *In Computer Network Security, ser. Lecture Notes in Computer Science, I. Kotenko and V. Skormin, Eds. Springer Berlin Heidelberg,* 7531: 240–253, 2012.

[6] P. Faruki, A. Bharmal, V. Laxmi, V. Ganmoor, M. Gaur, M. Conti, and R. Muttukrishnan. Android security: A survey of issues, malware penetration and defenses, 2015.

[7] Andi Fitriah A.Kadir, Natalia Stakhanova, Ali A. Ghorbani. Android Botnet: What URLs are telling us. *9th International Conference on Network and System Security (NSS), November 3-5,* 2015.

[8] P. Faruki, V. Ganmoor, V. Laxmi, M.S. Gaur and A. Bharmal. "AndroSimilar: robust statistical feature signature for Android malware detection." *In Proceedings of the 6th International Conference on Security of Information and Networks,* pp. 152-159. ACM, 2013.

[9] A.P. Fuchs, A. Chaudhuri, and J.S. Foster. Scandroid: Automated security certification of android applications. https://www.cs.umd.edu/ avik/papers/scandroidascaa.pdf

[10] W. Enck, P. Gilbert, B.G. Chun, L. P.Cox, J. Jung, P. McDaniel, and A. N.Sheth. TaintDroid: An Information-flow Tracking System for Realtime Privacy Monitoring on Smartphones. *In Proc. of the 9th USENIX Symposium on Operating Systems Design and Implementation,* USENIX OSDI '10, 2010.

[11] DROIDKIN Gonzalez, Hugo, Natalia Stakhanova, and A. Ghorbani. Droidkin: Lightweight detection of android apps similarity. *Proceedings of the 10th SECURECOMM,* 2014.

[12] M. Grace, Y. Zhou, Q. Zhang, S. Zhou, and X. Jiang. Riskranker: Scalable and accurate zero-day android malware detection. *In International conference on Mobile systems, applications, and services,* ACM, June 2012.

[13] M. Lindorfer, M. Neugschwandtner, L. Weichselbaum, Y. Fratantonio, V. van der Veen, and C. Platzer. Andrubis - 1,000,000 Apps Later: A View on Current Android Malware Behaviors. *In Proc. of the the 3rd International Workshop on Building Analysis Datasets and Gathering Experience Returns for Security, (BADGERS),* 2014.

[14] B. Sanz, I. Santos, C. Laorden, X. Ugarte-Pedrero, P.G. Bringas. PUMA: Permission usage to detect malware in android. *In Proc. of International Joint Conference CISIS'12-ICEUTE' 12-SOCO' 12 Special Sessions.* 2012.

[15] Asaf Shabtai Uri Kanonov , Yuval Elovici , Chanan Glezer , Yael Weiss, "Andromaly": a behavioral malware detection framework for android devices. *Journal of Intelligent Information Systems,* 38: 161–190, February 2012.

[16] Symantec, "latest intelligence for march 2016," in Symantec official Blog, 2016.

[17] K. Tam, K. Salahuddin, A. Fattori, and L. Cavallaro. Copperdroid: Automatic reconstruction of android malware behaviors, *In Proc. of the 2015 Network and Distributed System Security Symposium (NDSS'15), San Diego, CA, USA,* The Internet Society, February 2015.

[18] J. Oberhide. DISSECTING THE ANDROID BOUNCER. http://jon.oberhide.org/blog/2012/06/21/dissecting-the-android-bouncer/

[19] D.-J. Wu, C.-H. Mao and T.-E. Wei. Droid mat: Android malware detection through manifest and API calls tracing. *Seventh Asia Joint Conference on Information Security,* pp. 62-69, 2012.

[20] W.-C. Wu and S.-H. Hung. DroidDolphin: A Dynamic Android Malware Detection Framework Using Big Data and Machine Learning. *Conference on Research in Adaptive and Convergent Systems (RACS),* 2014.

[21] Rubin Xu , Hassen Saïdi , Ross Anderson, Aurasium: practical policy enforcement for Android applications. *In proceedings of the 21st USENIX conference on Security symposium,* p.27-27, August 08-10, 2012, Bellevue, WA.

[22] M. Zhao, F.Ge, T. Zhang and Z.Yuan. Antimaldroid: An efficient SVM-based malware detection framework for android. *Commun. Comput. Inf. Sci.* 243:158-166, 2011.

[23] Y. Zhou and X. Jiang. Android Malware Genome Project. http://www.malgenomeproject.org/.

[24] Y. Zhou, X.Zhang, X.Jiang and V.W. Freeh. Taming information- stealing smartphone applications (on android). *In Proc. of the 4th International Conference on Trust and Trustworthy Computing,* TRUST '11, 2011.

[25] Y. Zhou, Z. Wang, W. Zhou, and X. Jiang. Hey, you, get off of my market: Detecting malicious apps in official and alternative Android markets. *In Proc. of the 16th Network and Distributed System Security Symposium,* NDSS '12, February 2012.

[26] http://www.androidauthority.com/android-posts-highest-ever-market-share-figures-711517/

[27] https://file.gdatasoftware.com/web/en/documents/whitepaper/G_DATA_Mobile_Malware_Report_H1_2016_EN.pdf

[28] http://www.statista.com/statistics/271774/share-of-android-platforms-on-mobile-devices-with-android-os/

[29] http://www.statista.com/statistics/276623/number-of-apps-available-in-leading-app-stores/

[30] https://developer.android.com/training/permissions/requesting.html

[31] https://developer.android.com/guide/topics/security/permissions.html

[32] https://developer.android.com/guide/topics/security/permissions.html ♯ normal-dangerous

[33] http://sanddroid.xjtu.edu.cn:8080/
[34] http://www.appchina.com/
[35] http://apk.hiapk.com/
[36] http://android.d.cn/
[37] http://www.mumayi.com/
[38] http://apk.gfan.com/
[39] http://download.pandaapp.com/?app=soft &controller=android#.V-p3f4h97IU
[40] http://slideme.org/
[41] http://www.bluestacks.com/
[42] http://www.cs.waikato.ac.nz/ml/weka/