Technological University Dublin
ARROW@TU Dublin

Conference papers | School of Computer Sciences

2017

Dynamic behavior analysis of android applications for malware detection

Latika Singh

Markus Hofmann

Follow this and additional works at: https://arrow.tudublin.ie/scschcomcon

Part of the Computer Sciences Commons

This Conference Paper is brought to you for free and open access by the School of Computer Sciences at ARROW@TU Dublin. It has been accepted for inclusion in Conference papers by an authorized administrator of ARROW@TU Dublin. For more information, please contact arrow.admin@tudublin.ie, aisling.coyne@tudublin.ie, gerard.connolly@tudublin.ie.

This work is licensed under a Creative Commons Attribution-Noncommercial-Share Alike 4.0 License

2017 International Conference on Intelligent Communication and Computational Techniques (ICCT)
Manipal University Jaipur, Dec 22-23, 2017

Dynamic Behavior Analysis of Android Applications for Malware Detection

Latika Singh and Markus Hofmann
ITB Ireland
latikasingh@ncuindia.edu
markus.hofmann@itb.ie

Abstract: **Android is most popular operating system for smartphones and small devices with 86.6% market share (Chau 2016). Its open source nature makes it more prone to attacks creating a need for malware analysis. Main approaches for detecting malware intents of mobile applications are based on either static analysis or dynamic analysis. In static analysis, apps are inspected for suspicious patterns of code to identify malicious segments. However, several obfuscation techniques are available to provide a guard against such analysis. The dynamic analysis on the other hand is a behavior-based detection method that involves investigating the run-time behavior of the suspicious app to uncover malware. The present study extracts the system call behavior of 216 malicious apps and 278 normal apps to construct a feature vector for training a classifier. Seven data classification techniques including decision tree, random forest, gradient boosting trees, k-NN, Artificial Neural Network, Support Vector Machine and deep learning were applied on this dataset. Three feature ranking techniques were usedto select appropriate features from the set of 337 attributes (system calls). These techniques of feature ranking included information gain, Chi-square statistic and correlation analysis by determining weights of the features. After discarding select features with low ranks the performances of the classifiers were measured using accuracy and recall. Experiments show that Support Vector Machines (SVM) after selecting features through correlation analysis outperformed other techniques where an accuracy of 97.16% is achieved with recall 99.54% (for malicious apps). The study also contributes by identifying the set of systems calls that are crucial in identifying malicious intent of android apps.**

Keywords: Android malware detection, predictive, analytics

1. INTRODUCTION

In the present digital era, smartphones have become an essential part of our daily lives. According to Gartner(Release 2017), the mobile usage has reached 90 %penetration in the regions of America, Europe, Japan and Asia/Pacific. A wide range of services are provided through applications (apps) of smartphones including games, social media,banking, etc. On one hand, it is very convenient to have all these services through a small portable device, however, on the other hand, it carries considerable risk to have these on the web, as our personal details are vulnerable to attacks. Since the last decade, several malware authors have reportedly started writing apps for stealing crucial information (Mobile Security) (F-Secure, 2014).

At present, Android is the most popular operating system amongst all the available mobile-devices. International Data Corporation reported that Android has a market share of 86.6% (in 2016) in global market (Chau). This is mainly due to its open source nature and the availability of free apps on official as well as third party markets. However, its open source nature makes it more prone to attacks through apps with embedded malwares. As per reports, 99% of the mobile device attacks are on Android(F secure). Due to gravity of this problem, several researchgroups are working on designing systems that can detect the malicious apps to make Android safer for users. Most of the newly proposed systems are based on analyzing the static or dynamic features of apps. In static analysis, code of an app is analyzed without executing it with the aim to identify malicious segments. Whereas this approach is quick, itoften fails against code obfuscation in which code is transformed into polymorphic form to avoid its reverse engineering. To overcome this problem, researchers have thought of capturing and modeling the run-time behavior of apps. In these approaches,various features of running apps are extracted which are then used to train classifiers. (Ham & Choi, 2013) used a feature set consisting of Network data, SMS, CPU utilization, power consumed, memory occupied by libraries and virtual memory utilization to train Naïve Bayes, Logistic Regression, Random Forest and SVM. It was found that Random forest algorithm outperformed the other algorithms in detecting the apps with malware intent. However, very few malware samples were taken in this study and the number of normal apps was much higher than the malware ones that might have led to class imbalance problems. Also the feature set used was taken assuming the malware apps are resource exhaustive which might also be characteristic properties of benign apps. In a similar study Lu, et al(2013), collected runtime behavior features like use of permission to change the network state, send SMS, etc. Chi-square test was then applied for feature selection. The selected features served as input to Bayesian method for classification which classified the samples with 89% accuracy. The study can be extended by using more feature selection approaches and supervised learning algorithms to improve the efficiency of detection. Tenenboim-Chekina, et al.(2013) had collected network related features of apps in execution at regular intervals to capture the details of upgrades by these apps. The correlation between these features was also calculated to detect the

978-1-5386-3030-3/17/$31.00 ©2017 IEEE

Authorized licensed use limited to: Technological University Dublin. Downloaded on January 20,2022 at 16:19:19 UTC from IEEE Xplore. Restrictions apply.

abnormalactivities. This was referred to as cross feature analysis.They were able to detect the repacked malicious app; however, the details of number and type of samples were not provided in the paper, making it difficult to analyse their work. Also, more can be done to improve this work by examining higher order statistics of the features collected. In 2013,Alam&Vuong(2013) extracted runtime features related to battery, binder, memory, CPU, network and permissions. Training Random Forest classification model yielded 99% accuracy. The details about falsepositives are not mentioned in the paper.. A study (iMas'ud, Sahib, Abdollah, Selamat, &Yusof, 2014)was conducted where different feature selection methods were applied before applying five different machine learning algorithms. They applied Chi-square and information gain for selecting the features before training the classifiers: Naïve Bayes, K-NN, Decision Tree, Multi-layer perceptron and Random Forest. The best results were obtained from neural network model after feature selection where accuracy of 83% was achieved. This study can be extended by applying deep learning method which might provide better accuracy. (Ng & Hwang, 2014) demonstrated that Dendritic Cell algorithm is better for classifying the normal and malicious app. They extracted system call behavior of the apps while the apps were running. However, the dataset used for this study was not sufficient and comparison with other algorithms on the same dataset was not provided. In a study conducted by Kim & Choi in 2014 extracted memory, CPU and Network related features and applied feature selection approach in which onefeature was removed and performance of the classifier was measured. The best performance of 95.97% was achieved after removing 23 features. However, this way of removing the feature is not very useful as some of the features might be good when they are considered with other features and the joint statistics couldhave led to better results. A similar study that investigated the anomalies of features at execution time like power consumption, network traffic and battery temperature was done by Kurniawan, Rosmansyah, &Dabarsyah in 2015.They applied four machine learning algorithms including J48, Random Forest, SVM and LMT on combination of three types of features. Results indicate that batterytemperature was not contributing much to the detection and with the remaining two features J48 outperformed the other machine learning algorithm.

The present study tries to fill the gaps discussed in the mentioned studies. We have extracted system calls invoked by normal and malicious apps duringexecution. The system calls were chosen as features since all the resources are ultimately allocated by the Linux kernel to the apps through a set of system calls. Counting particular system call requests, allows us to estimate the resource utilization of the app.

2. DATASET

For behavior analysis, 278 non-maliciousapps and 216 malicious apps were used. The normal apps were taken from the Google Play Store whereas the malicious appswere taken from the contagio project (Parkour, 2016).These apps were installed in the emulator (API Level 16, version 4.1.1) using *adb*(Android Debug Bridge)install command.All the apps were executed using the *monkey*tool(UI/Application Exerciser Monkey)which simulates the usage of the application and is generally used for stress testing of the applications being developed. System call behavior (337 system calls of Linux) of each app was monitored using an automatic script that was written as a shell script. The data was retrieved from the emulator shell using adb tool and a Python script was written to format the dataset. The feature vector consisted of 337 attributes corresponding to each system call. The value of the attribute was the number of times that particular system call was invoked during the execution of the app.

3. METHODOLOGY

Once the feature set was constructed, the classifier models were trained and validated. To improve the performance of these classifiers, feature ranking and selection techniques were also applied and the performance before and after application were compared. The block diagram of the process followed is shown in Figure 1.

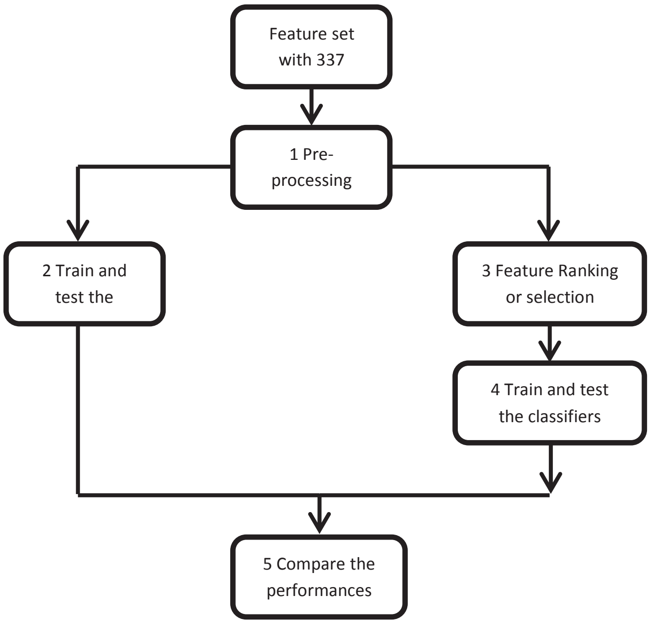

Figure 1 Block diagram of the process

2.1 ***Pre-processing:-*** During this step, attributes with zero variance were removed. These features correspond to the system calls that were never invoked by any app of the sample set. Out of the original 337 system calls 43 attributes (excluding nominal class label) were selected. The parallel plot of these 43 attributes shown in Figure 2 indicates that some of these attributes have predictive power

2.2 ***Classification:*** It is a process in which a training data set with input and output pairs is analyzed by

Authorized licensed use limited to: Technological University Dublin. Downloaded on January 20,2022 at 16:19:19 UTC from IEEE Xplore. Restrictions apply.

the algorithm to learn a mapping function which can later be used for predicting the unknown outputs for some input data. In the present case, the input attributes are the

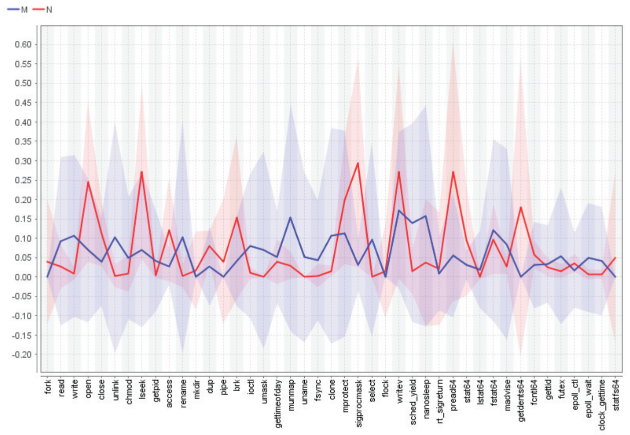

Figure 2 Parallel plot of 43 features for 2 output categories (red – normal, blue – malicious)

frequencies of system calls invoked and the output variable is the type of app (benign or malicious). Once the mapping function is inferred we can monitor the system call behavior of unknown apps and judge whether these have malicious intent or not. Seven machine learning algorithms were applied,namely Decision Trees, K-nearest Neighbors (K-NN), Support Vector Machine, Neural Network and Deep Learning. These are briefly described as follows:

***3.1Decision Tree Classification**:* This classification algorithm builds the model in the form of a decision tree which is a tree-like graph where internal nodes are attributes; branches denote values that satisfy some conditions or tests and leaf nodes are class labels (Quinlan, 1986). The decision tree is constructed using a greedy algorithm in which the tree is built recursively by choosing appropriate attribute as root of the tree(Rokach & Maimon, 2008). The choice of attribute is done by finding optimal splitting condition such that after applying this condition on the selected attribute, the resulting partitions are as pure as possible.

3.2 Random Forest :are also referred as random decision forests. Random Forest classification is an ensemble learning method for classification and is based on decision tree learning(Ho, 1995). Decision tree learning suffers the drawback of over-fitting where the trees with deeper levels represent irregular patterns as they overfit the training dataset. To overcome this problem, random forest classification averages multiple decision trees that are trained on different parts of the same training dataset(Breiman, 2001). The idea is to reduce the variance that is observed in trees that have grown very deep. This process substantially enhances the performance of the model and is a part of bagging approach where random samples of data are taken to train different trees which are later averaged to find better performing model.

3.3 Gradient Boosted Trees:Boosting is similar to bagging except that the instead of training the trees with random training data, the trees are grown to weighted versions of the dataset where more weights are given to observations that are harder to learn(Freidman, 1999). Due to weighting, the trees are de-correlated by focusing on regions that are missed by the previously trained trees.

3.4k-NN (K nearest neighborhood): is a lazy learner classifier in which the process of modeling is delayed until it is needed to classify the given input sample(Coomans & Massart, 1982). In this method the data are dividedinto test data and training data. For each sample row of test data, *k* nearest neighbors are determined by computing the proximity of the test tuple with the rest of the tuples or data points using distance functions like Euclidean distance. The output class of the test rowis then assigned the value of majority class of these neighbors.

3.5 Support Vector Machine (SVM) : is a classification technique based on statistical learning theory. The Linear SVM identifies hyper-plane with maximum margin to separate the two classes(Press, Teukolsky, Vetterling, & Flannery, 2007).

3.6 Artificial Neural Networks:are inspired by the biological neural system. It is composed of nodes and connections(Minsky & Papert, 1969). The network will have an input layer, intermediary hidden layers and an output layer. In a feed-forward neural network, the nodes in one layer are connected to nodes of the next layer. The connections have some weights that are randomly assigned atthe beginning and slowly learnt based on the training data(Artifical Neural Network 2017). Each node calculates the output based on the weighted input received and its activation function. The final outputs are then matched with the expected outputs and are errors are calculated. The errors are propagated back and the weights are adjusted. This is known as back-propagation.

3.7 Deep Learning: is based on a multi-layer feed-forward artificial neural network that is trained with stochastic gradient descent using back-propagation. In this cascade of many layers of neurons (processing units) is used for feature extraction and transformation. More layers as compared to other leaners are used in deep learning. At each layer the parameters of inputs are learned and transformed. It is based on distributed representations.(Deng & Yu, 2014).

3.8 Feature Ranking and Selection: The performance of classifiers can improve if we relieve the assumption that all the features are of equal importance for predicting the class label (Kolcz & Teo, 2009). The feature can be assigned some weight based on criteria like information gain ratio, Chi-square statistics, etc. The higher the weight of an attribute the more relevant it becomes for predicting the class label. This paper examines the impact of three feature selection techniques based on weighting techniques on performance of

Authorized licensed use limited to: Technological University Dublin. Downloaded on January 20,2022 at 16:19:19 UTC from IEEE Xplore. Restrictions apply.

the classifiers, namely information gain ratio, Chi-square statistics and correlation. These are explained as follows

***3.8.1 Feature weighting using information gain*:** The attributes including the output class are taken as random variables. The information gain is determined by knowing the presence and absence of an attribute with the aim to find out how much information gain is achieved by adding an attribute in the input feature set that is used for training the model (Mladenic, Brank, Grobelnik, & Milic-Frayling, 2004). This can be calculated by using information-theoretic definition. Assuming C is output class (label) and K is one of the predictor attribute (both are considered random variables) then information gain can be defined as

$$IG(t) = H(C) - H(C|K) \quad (1)$$

3.8.2 Feature weighting using Chi-square statistic: The Chi-square test is a statistical test used to determine the independence or dependence between two variables. This can be applied in feature selection as we can determine the dependence of each input attribute and target output class and weight/rank the input attributes accordingly. The attribute which is more dependent is given higher weight and the attribute which is not dependent is discarded. For continuous variables or numerical data the Chi-square is applied after binning the values. The value of the Chi-square is

$$x^2 = \sum_{i=1}^{n} \frac{(O_i - E_i)^2}{E_i} \quad (2)$$

Where x^2 = Pearson's cumulative test static, *Oi* = number of observations of type *I, Ei*= expected frequency of type *I, and n*=number of rows in the table

3.8.3 Feature weighting using correlation:A feature is considered useful if it is correlated well with category membership. In this technique we try to find a subset that contains features that are highly correlated with the output label and un-correlated with each other. The following equation finds the merit of a feature subset D consisting of m features:

$$Merit_{D_m} = \frac{m\overline{r_{cf}}}{\sqrt{m + m(m-1)\overline{r_{ff}}}} \quad (3)$$

Where $\overline{r_{cf}}$ is the average value of all feature-classification correlations and $\overline{r_{ff}}$ is the average value of all feature-feature correlations.

4. PERFORMANCE MEASURES

Evaluation of the performance of classifier is based on the number of samples that are correctly or incorrectly categorized by the classification model. The measures used in this paper are accuracy, recall and precision.

4.1 Accuracy: Accuracy is a measure of how many of the total instances are correctly predicted by the model. It is defined by the equation

$$Accuracy = \frac{Number of correct predictions}{Total number of predictions}$$

4.2 Precision and Recall: The accuracy measures do not treat the class differently and more parameters are required to analyse datasets with imbalance. Usually, the rare class (for example malware) is more interesting than the majority class (normal apps). Precision refers to the fraction of the examples that are actually positive in the group that the classifier has predicted as a positive class. In our paper, precision indicates how many apps were actually malicious out of the predicted ones. Higher value of precision means lower false positive rates. Recall, on the other hand, measures the fraction of positive examples that are correctly predicted as positive by the classifier. For our study this means that out of all the malicious apps how many were correctly predicted as malicious. In this paper it is more important to have better recall (for malicious class) than for the non-malicious class because the risk of categorizing malicious app as normal is more dangerous than identify normal as malicious (false positive).

5. EXPERIMENTAL RESULTS

A 10-fold cross validation using stratified sampling was applied, creating 10 mutually exclusive subgroups each used for training and testing. . The results of all the classifiers before applying any feature select are presented in Table 1.

TABLE 1 PERFORMANCE OF CLASSIFIERS BEFORE APPLYING FEATURE SELECTION

Classifier	Parameter setting	Accuracy	Precision	Recall
Decision Tree	Gini Index for splitting	98.26% +/- 1.58%	98.57%	95.96%
Random Forest	30 trees	96.77% +/- 2.37%	99.23%	90.45%
Gradient Boosting Trees	20 trees	98.50% +/- 1.66%	98.52%	96.73%
K-NN	K=1	94.29% +/- 2.73%	94.28%	87.31%
Support Vector Machine	Anova kernel	96.51% +/- 3.39%	97.63%	91.15%
Neural Network	Epocs=1000	94.07% +/- 3.31%	89.51%	92.95%
Deep Learning	Epocs=15	96.03% +/- 2.52%	94.72%	92.69%

It is evident from Table 1 that the Gradient Boosting Trees algorithm is giving the best accuracy amongst all the methods. The next section provides the performance of these classifiers after applying the feature selection algorithms.

Authorized licensed use limited to: Technological University Dublin. Downloaded on January 20,2022 at 16:19:19 UTC from IEEE Xplore. Restrictions apply.

5.1 Performance after feature selection using information gain

The normalized weights of the features were calculated using information gain. The features having weights less than 0.05 are rejected, which led to improvement in performance of the classifiers. Rejecting features beyond this was leading to a reduction in performance of the classifier, thus 0.05 was taken as optimal threshold. The parameters of the models mentioned in Table 2 were not changed. Three attributes were rejected during this process; these are rt_sigreturn, flock and mkdir. This new subset of features was used to train the classifiers again and the performances were measured (see Table 2). The comparative performances are shown in Figure 3. In this figure the suffix –p is used with name of the classifier to denote the classifier performance before applying the feature selection. For example supportvector machine-p and support vector machine correspond to performance of support-vector machine before and after applying the feature selection respectively.

TABLE 2 PERFORMANCE OF CLASSIFIERS AFTER FEATURE SELECTION USING INFORMATION GAIN

Classifier	Accuracy	Precision	Recall
Decision Tree	97.64% +/- 2.06%	98.57%	95.83%
Random Forest	94.56% +/- 3.47%	91.6%	96.30%
Gradient Boosting Trees	98.38% +/- 1.51%	99.06%	97.22%
K-NN	96.56% +/- 4.24%	95.85%	96.3%
Support Vector Machine	96.76% +/- 2.27%	93.48%	99.54%
Neural Network	95.75% +/- 2.30%	94.12%	96.30%
Deep Learning	97.17% +/- 1.34%	98.13%	97.22%

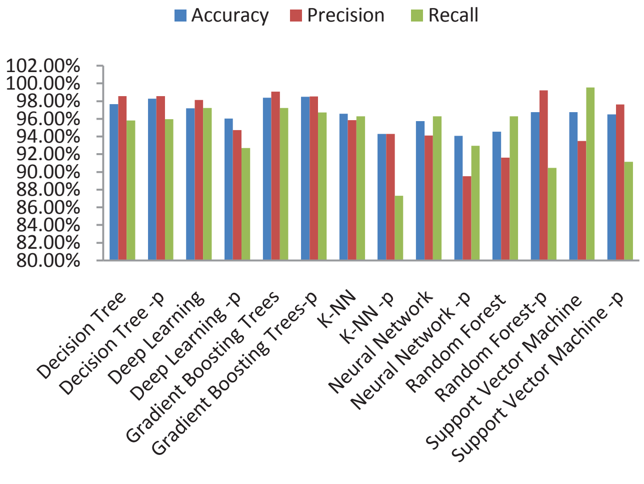

Figure 3- Comparison in performance after feature selection using information gain

Substantial improvement in recall of k-NN, Deep learning, SVM, Random forest and neural network can be seen. This is very favorable to the objective of the study as it is important to have a good recall (for malicious class) that may come at cost of misclassifying some of the normal apps as malicious. In the next section, performance of classifiers after feature selection using Chi-square statics are presented.

5.2 Performance after feature selection using Chi-square statistics

The Chi-square statistics is used to determine weight of attributes where the highest weight is given to the most relevant attribute. The attributes having a weight less than 0.05 were rejected; using this threshold six attributes were rejected, namely,rt_signreturn, flock, mkdir, lstat64, statfs64, epoll_ctl. Out of these six, three attributes were also recommended for rejection using the information gain approach. The classifiers were trained using the feature subset and performances were measured. The results are described in Table 3.

TABLE 3 PERFORMANCE OF CLASSIFIERS AFTER FEATURE SELECTION USING CHI-SQUARE STATISTICS

Classifier	Accuracy	Precision	Recall
Decision Tree	97.17%+/- 2.06%	97.64%	95.83%
Random Forest	95.15% +/- 2.73%	92.48%	96.76%
Gradient Boosting Trees	98.38% +/- 1.51%	99.06%	97.22%
K-NN	96.56% +/- 4.24%	95.85%	96.30%
Support Vector Machine	96.96% +/- 2.28%	93.89%	99.54%
Neural Network	95.15% +/- 3.76%	92.48%	96.76%
Deep Learning	96.97% +/- 1.00%	97.18%	95.83%

Similar results were obtained where recall (corresponding to malicious class) was improved in all models except decision trees.

5.3 Performance Feature selection using correlation

Finally, feature selection was performed by calculating the ranks of the attributes using the correlation approach which attemptsto find a subset of features that are highly correlated with the output class and least correlated with each other. In this case, the threshold for selection was 0.2 which means that attributes having less than 0.2 weight (normalized) were rejected. Using this approach 12 attributes were rejected, namely, rt_sigreturn, flock, mkdir, gettid, gettimeofday, fstat64, lstat64, epoll_ctl, statfs64, fork, pipe and futex. The performances of classifiers trained on this feature subset are mentioned in Table 4.

Authorized licensed use limited to: Technological University Dublin. Downloaded on January 20,2022 at 16:19:19 UTC from IEEE Xplore. Restrictions apply.

TABLE 4 PERFORMANCE OF CLASSIFIERS TRAINED ON FEATURE SUBSET SELECTED USING CORRELATION APPROACH

Classifier	Accuracy	Precision	Recall
Decision Tree	97.58% +/- 2.34%	97.22%	97.22%
Random Forest	96.56% +/- 1.58%	95.02%	97.22%
Gradient Boosting Trees	99.19% +/- 0.99%	99.07%	99.07%
K-NN	98.79% +/- 2.05%	98.61%	98.61%
Support Vector Machine	97.16% +/- 1.63%	94.30%	99.54%
Neural Network	93.13% +/- 4.50%	93.33%	90.74%
Deep Learning	97.17% +/- 1.34%	96.33%	97.22%

With all three feature subsets, the best recall for malicious class was obtained using a support vector machine classifier. However, this is true only when a suitable subset of feature was selected. The performance of SVM with all the three feature subsets is shown in Figure 4.

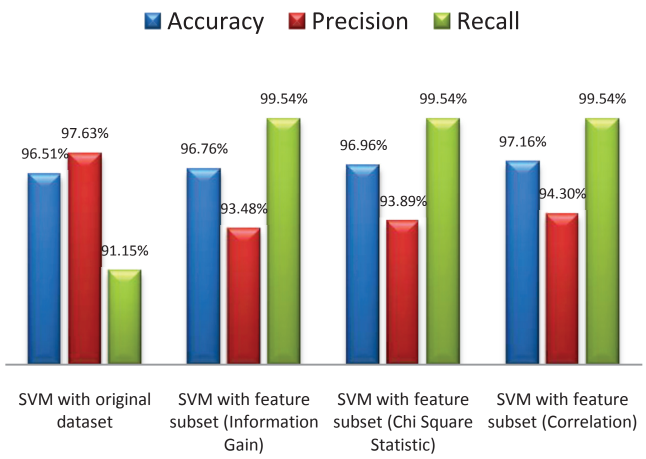

Figure 4 Comparison of SVM performances with various feature sets

Since recall for malicious class with SVM in all the three feature subsets is equal, the accuracy and precision can be used to evaluate the performance of the right feature subset. In conclusion, we can state that SVM with feature selection using correlation marginally outperformed all other classifier and feature selection techniques.
In summary, during the initial screening of 337 systems calls as features, 294 features were discarded as these system calls were never invoked by the 494 apps collected in this study. The remaining 43 features were used to train the classifier and anacceptable accuracy was achieved. However, to improve the performance further, three features techniques were applied. A maximum of 12 features were removed by feature selection using the correlation technique (with 0.2 as threshold). This selected subset consisting of 31 features as a training set for support vector machine yielded very good class recall of 99.54%. We can therefore state that system calls are useful and can be monitored for identifying suspicious activities and therefore malicious mobile applications on Android device.

5. CONCLUSION AND FUTURE WORK

The present study was conducted to develop models that can identify the malicious intents of android apps using their run-time behavior. This dynamic behavior was measured by looking at the frequency of system calls made by a mobile app when it was running as process in the Linux kernel of the android operating system. After pre-processing and applying feature selection techniques it was found that 31 out of 337 system calls are excellent predictors of malicious apps. An accuracy of 97.16% and recall of 99.54% was achieved using a support vector machine classifier which performed better than decision trees, random forests, gradient boosted trees, neural network, k-NN and deep learning. Though the accuracy of gradient boosted tree was higher than SVM,the class recall was slightly lower which is more relevant for the problem under investigation. It is less risky to have a normal app being predicted as malicious than predicting a malicious one as normal. Most of the studies that have been conducted to analyse the dynamic behavior have looked at the statistics and usage of the resources like CPU time, network packets etc. In the present study we have examined the system call behavior because any resource be it CPU or network will be accessed through operating system system-calls. We have not come across any studies which have done comparative analysis of performances of various classification algorithms in predicting the malicious apps through system call behavior. Moreover the present study has identified a set of 31 system calls that are crucial in differentiating between normal and malicious apps (mentioned in Table 5).During the experiments one laptop on which emulator was installed crashed and an android device on which the experiments were conducted came under a ransom ware attack.
However, the study can be extended in several dimensions. The training set can be increased by collecting more malicious apps of various categories. The feature set can be made richer by adding static and dynamic features. More feature learning techniques can be explored such as evolutionary techniques particle optimization, or any colony optimization.

TABLE V LIST OF SYSTEM CALLS IMPORTANT IN IDENTIFYING MALWARE BEHAVIOR

S.No	Name of the system call	Function of the system call
1	Read	Read data from files/device
2	Write	Write data to device/files
3	Open	Open file
4	Close	Close file
5	Unlink	Delete files
6	Chmod	Change permission

Authorized licensed use limited to: Technological University Dublin. Downloaded on January 20,2022 at 16:19:19 UTC from IEEE Xplore. Restrictions apply.

7	Lseek	Change location of read/write pointer
8	Getpid	Get process identifier
9	Access	Check access to a file
10	Rename	Renames a file
11	Dup	Creates copy of file descriptor
12	Brk	Change the location of program break
13	Ioctl	Manipulate device parameters of special files
14	Umask	sets the calling process's file mode creation mask (umask) to mask & 0777
15	Munmap	deletes the mappings for the specified address range
16	Uname	returns system information
17	Fsync	synchronize a file's in-core state with storage
18	Clone	create a child process
19	Mprotect	set protection on a region of memory
20	Sigprocmask	examine and change blocked signals
21	Select	synchronous I/O multiplexing
22	Writev	write data into multiple buffers
23	Sched_yield	yield the processor
24	Nanosleep	high-resolution sleep
25	Pread64	read from a file descriptor at a given offset
26	Stat64	get file status
27	Madvise	give advice about use of memory
28	Getdents64	get directory entries
29	Fcntl64	manipulate file descriptor
30	Epoll_wait	wait for an I/O event on an epoll file descriptor
31	Clock_gettime	retrieve and set the time of the specified clock *clk_id*.

REFERENCES

[1] *Mobile Security*. (n.d.). Retrieved Dec 29, 2006, from Wikipedia: https://en.wikipedia.org/wiki/Mobile_security#cite_note-FOOTNOTESchmidtSchmidtBatyukClausen2009a3-27

[2] F-Secure. (2014). Mobile Threat Report.

[3] Chau, M. (n.d.). *Smart Phone OS Market Share 2016*. Retrieved Dec 30, 2016, from http://www.idc.com/promo/smartphone-market-share/os

[4] Ham, H.-S., & Choi, M.-J. (2013). Analysis of Android Malware Detection Performance using machine learning classifiers. *2013 International Conference on ICT Convergence (ICTC)* (pp. 490-495). IEEE.

[5] Lu, Y., Zulie, P., Jingju, L., & Yi, S. (2013). Android Malware Detection Technology Based on Improved Bayesian Classification. *3rd International conference on instrumentation, measurement, computer, communication and control* (pp. 1338-1341). IEEE

[6] Tenenboim-Chekina, L., Barad, O., Shabtai, A., Mimran, D., Rokach, L., Shapira, B., et al. (2013). Detecting Application Update Attack on Mobile Devices through Network Features. *Computer Communications Workshop INFOCOM* (pp. 91-92). IEEE.

[7] Alam, M. S., & Vuong, S. T. (2013). Random Forest Classification for Detecting Android Malware. *Green Computing and Communications* (pp. 663-669). IEEE.

[8] i Mas'ud, M. Z., Sahib, S., Abdollah, M. F., Selamat, S. R., & Yusof, R. (2014). Analysis of feature selection and machine learning classifier in android malware detection. *International conference on information science and application* (pp. 1-5). IEEE

[9] Ng, D. V., & Hwang, J.-I. G. (2014). Android malware detection using dendritic cell algorithm. *International Conference on Machine Learning and cybernetics* (pp. 257-262). Lanzhou: IEEE

[10] Kim, H.-H., & Choi, M.-J. (2014). Linux kernel-based feature selection for Android malware detection. *Asia Pacific Network Operation and Management Symposium* (pp. 1-4). IEEE.

[11] Kurniawan, H., Rosmansyah, Y., & Dabarsyah, B. (2015). Android anomaly detection system using machine learning classification. *International Conference on Electrical Engineering and Informatics* (pp. 288-293). IEEE.

[12] Parkour, M. (2016, July 9). *Contagio Mobile*. Retrieved March 2016, from http://contagiominidump.blogspot.in/

[13] *Android Debug Bridge*. (n.d.). Retrieved Dec 12, 2016, from Android Developers: https://developer.android.com/studio/command-line/adb.html

[14] *Artifical Neural Network*. (n.d.). Retrieved Dec 22, 2016, from Wikipedia: https://en.wikipedia.org/wiki/Artificial_neural_network

[15] Breiman, L. (2001). Random Forests. *Machine Learning, 45*(1), 5-32.

[16] Coomans, D., & Massart, D. L. (1982). Alternative k-nearest neighbour rules in supervised pattern recognition : Part 1. k-Nearest neighbour classification by using alternative voting rules. *Analytica Chimica Acta*, 15-27.

[17] Deng, L., & Yu, D. (2014). Deep Learning Methods and Applications. *Foundations and trends in signal processing*, 3-7.

[18] Freidman, J. H. (1999). Greedy Function Approximation: A Gradient Boosting Machine. http://www-stat.stanford.edu/~jhf/ftp/trebst.pdf.

[19] F-Secure. (2014). Mobile Threat Report.

[20] Ho, T. K. (1995). 3rd International Conference on Document Analysis and Recognition. Proceedings of the 3rd International Conference on Document Analysis and Recognition, Montreal, QC.

[21] Kolcz, A., & Teo, C. H. (2009). Feature weighting for improved classifier robustness. *Sixth conference on email and anti-spam*, (pp. 1-8). California.

[22] Minsky, M., & Papert, S. (1969). Perceptrons: An Introduction to Computational Geometry. MIT Press.

[23] Mladenic, D., Brank, J., Grobelnik, M., & Milic-Frayling, N. (2004). Feature selection using linear classifier weights: interaction with classification models. *SIGIR* (pp. 1-8). Sheffield: ACM.

[24] *Mobile Security*. (n.d.). Retrieved Dec 29, 2006, from Wikipedia: https://en.wikipedia.org/wiki/Mobile_security#cite_note-FOOTNOTESchmidtSchmidtBatyukClausen2009a3-27

[25] Press, W. H., Teukolsky, S. A., Vetterling, W. T., & Flannery, B. P. (2007). *Numerical Recipes: The Art of Scientific Computing*. New York: Cambridge University Press.

[26] Quinlan, J. R. (1986). *Induction of Decision Trees. Machine Learning.* Kluwer Academic Publishers.

[27] Release, G. P. (n.d.). *Gartner Newsroom*. Retrieved Feb 28, 2017, from Gartner Newsroom: http://www.gartner.com/newsroom/id/3339019

[28] Rokach, L., & Maimon, O. (2008). *Data mining with decision trees: theory and applications*. World Scientific Pub Co Inc.

[29] *UI/Application Exerciser Monkey*. (n.d.). Retrieved Dec 12, 2016, from Android Developer: https://developer.android.com/studio/test/monkey.html

Authorized licensed use limited to: Technological University Dublin. Downloaded on January 20,2022 at 16:19:19 UTC from IEEE Xplore. Restrictions apply.