An Automated Malware Detection System for Android using Behavior-based Analysis AMDA

Abela, Kevin Joshua L.
College of Computer Studies
De La Salle University – Manila
1004 Manila, Philippines (632) 524-4611 loc. 130
kevin.abela@live.com

Angeles, Don Kristopher E.
College of Computer Studies
De La Salle University – Manila
1004 Manila, Philippines (632) 524-4611 loc. 130
donangeles@live.com

Delas Alas, Jan Raynier P.
College of Computer Studies
De La Salle University – Manila
1004 Manila, Philippines (632) 524-4611 loc. 130
jandelasalas@live.com

Tolentino, Robert Joseph
College of Computer Studies
De La Salle University – Manila
1004 Manila, Philippines (632) 524-4611 loc. 130
robert.tolentino@live.com

Gomez, Miguel Alberto N.
College of Computer Studies
De La Salle University – Manila
1004 Manila, Philippines(632) 524-4611 loc. 130
miko.gomez@delasalle.ph

***Abstract*— The Android platform is the fastest growing market in smartphone operating systems to date. As such, it has become the most viable target of security threats. The reliance of the Android Market Security Model on its reactive anti-malware system presents an opportunity for malware to be present in the Official Android Market and does not encompass applications outside the official market. This allows applications to masquerade as harmless applications which lead to the loss of credentials if precautions are not taken. Most anti-malware applications in the Market use static analysis for detection because it is fast and relatively simple. However, static analysis requires regular updates of threat databases and it may be circumvented by obfuscation techniques. As a solution to these problems, the study utilizes behavior analysis of applications as basis for malware. As a first step, features of known-benign and known-malicious applications are extracted for machine learning to provide baseline behavior datasets. Test applications are then passed through the behavior based module for identification of its being malware or benign. A classification scheme is provided for applications identified as malware by the system.**

Keywords—Android, Security, Behavior Analysis

1 INTRODUCTION

Smartphones are now a target for malicious software which attempt to damage personal assets of users. The Android platform, being the fastest growing market today, faces the same risk. Malware takes advantage of the platform for its being open, complete and free for development meaning that there is lack in control for application development. Allowing anyone to develop and to publish applications into the Android Market presents an opportunity for attackers to easily deliver malicious applications onto unsuspecting users. 1 The presence of alternative Android market makes this problem worse because of the lack in review methods thus making them unreliable sources of applications.

The Android platform uses permissions-based security models to have access to different functionalities of devices. This model provides information about the access and privilege capacity of an application which to more technical users, may be used as an indication for malicious intent but to normal users, this information is often neglected thus making this model unreliable on its own. Static analysis is a method used to detect malware but this method proves to be inadequate as malware can be undetected by obfuscation, as an example. The time it takes to manually check the code provides an opportunity for malware to infect devices before they are detected.

To address these problems, an automated behavioral analysis system called AMDA is the solution. The AMDA system determines malicious behavior from benign behavior through the use of machine learning techniques. A behavior model for trojans, spyware, viruses and exploits are generated and used for classification of applications. Results are verified by forwarding them to an expert system, VirusTotal.

2 AMDA

AMDA is an automated malware detection system for the Android platform. This paper includes the discussion of the core modules of the system namely the Feature Extraction Module and the Behavior Analysis Module. These modules are responsible for behavioral analysis to detect malicious activity of applications based on extracted features. The current development of the system involves categorization of applications based on the AMDA's classification and cross-validation of the results to the expert system, which generally concludes the functionalities of the system.

A. *Applictaion Acquisition Module*

The Application Acquisition module is responsible for downloading applications from Android Markets and storing them into the application repository inside the server. Test applications are downloaded both from the Official Android market and some Alternative Android markets. Benign applications are downloaded from the Official Android Market and known-malware applications are downloaded from online Android malware providers such as VirusTotal and Contagio. For the downloading of test applications, a web-crawler tool is used applications from alternative Android market domains which provide free download of .apk files. The applications are forwarded to the VirusTotal Malware Verification System (VMS) to be able to acquire the classification of the test applications for use in later modules. As for downloading the training applications, these are acquired manually in order to ensure the validity of the applications.

Applications from the Official Android Market and the Android malware providers are acquired manually. The downloaded applications are also tested for their validity as benign or as known-malware through the VirusTotal VMS. Once the status of each application is confirmed, the application is passed on to the repository to be used by the next module, the Feature Extraction module.

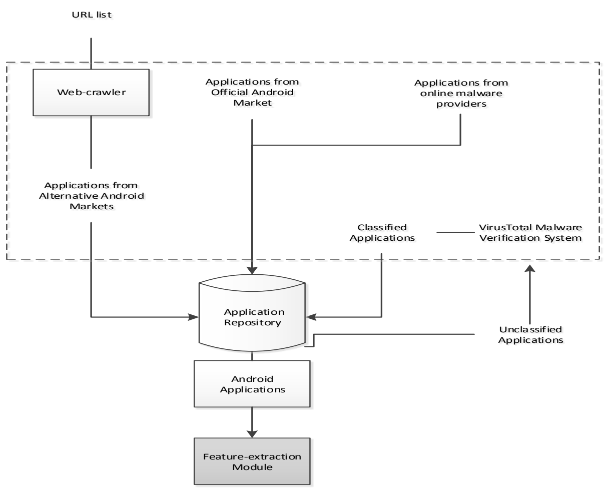

URL list
Web-crawler
Applications from Official Android Market
Applications from online malware providers
Applications from Alternative Android Markets
Classified Applications
VirusTotal Malware Verification System
Application Repository
Unclassified Applications
Android Applications
Feature-extraction Module

Figure 1. Application Acquisition Module

1) *Web-crawler Submodule*

The web-crawler iterates selected sites from a predefined list of domains to search for free Android applications in .apk format. When an Android application has been found, the web-crawler downloads the file and stores it in the applications repository. When the web-crawler has finished all the URLs in the list, it forwards the applications to the VirusTotal VMS for classification. Once all the applications have been classified, the process of crawling is repeated starting from the first defined domain. The process may take some time to finish depending on the user's Internet connection and is best done at least daily to check gather newly uploaded applications from the alternative Android markets.

B. *Feature Extraction Module*

The Feature Extraction Module is the one that generates activity log from running applications retrieved from the application repository of the system. The activity log contains the system calls from application activity which are the features that the module retrieves. For these features to be extracted, the logs are processed by the Virtualization Submodule which handles monitoring and logging of application activity. Features acquired from the Virtualization Submodule are filtered through parsing before being forwarded to the Behavior-based Analysis Module.

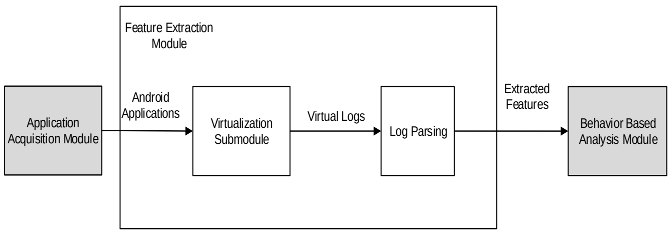

Feature Extraction Module
Application Acquisition Module
Android Applications
Virtualization Submodule
Virtual Logs
Log Parsing
Extracted Features
Behavior Based Analysis Module

Figure 1. Feature Extraction Module

1) *Virtualization Submodule*

An Android 2.3.3 SDK emulator is used to run the Android applications because this is the only medium to automate the generation of application system activity logs without using an actual mobile device. In order to automate the extraction of the system calls generated by the applications, the Android SDK emulator is run together with Monkey, a tool which simulates user input. According to 2, there is no actual difference to using human input to be able to activate the malicious activity of an application.

The collection of system calls is done through the use of Strace. The tool monitors and logs low-level activity in kernel space in the Android SDK emulator. This method of monitoring low-level system calls ensure that all activity of the application is recorded. 3 However, the log data contains activity which are irrelevant for detection of malicious activity. With this problem of noise in the log data, the system utilizes a self-developed parser which can be customized as to which features are to be collected.

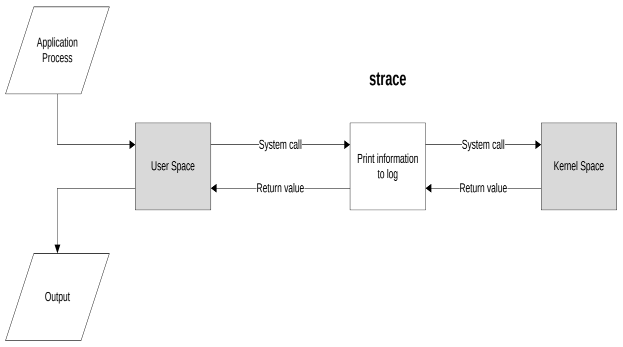


Figure 2. Virtualization Submodule

Features of the system are mapped to activities of the application which may indicate malicious activity. The following system calls are typically executed by Android Malware 34:

Table 1. Mapping of System Calls to Application Activity

Activity Monitored	System Calls
Incoming and outgoing network traffic in the application layer	Read(), write(), Brk(), getpid(), Sigprocmask()
Read and write operations on all storages	Read(), Write(), Recv(), lseek(), getpid()

Services and processes started	Open(), Msgget(), Close(), getpid(), Semget(), Semop(), Clone(), System_224()
File transfer through the network	Dup(), Fork()
Bypassed permissions	Ioctl(), mprotect()

Table 3. Detailed Description of System Calls [18]

System Calls	Definitions
Read()	*reads from file descriptors*
Write()	*writes to a file descriptors*
Fork() and Clone()	*creates a child process*
Lseek()	*repositions read and write offsets*
getpid() and System_224()	*gets process/ thread information/ identification*
Dup()	*Duplicates open file descriptors*
Ioctl()	*controls input/output devices*
Clone()	*creates child process*
Sigprocmask()	*examine and change blocked signals*
mprotect()	*changes access protections for the process memory pages*
Semget()	*Returns the semaphore indentifier associated with the given key*
Semop()	*Used in semaphore operations such as signalling and waiting*
Brk()	*change the amount of space allocated for a process*
Recv()	*Receive message from a socket.*
Open()	*Returns a file descriptor*
Close()	*Closes a file descriptor*
Msgget()	*Creates or return results from a message queue.*

C. *Behavior-based Analysis Module*

The Behavior-based Analysis Module is responsible for classifying Android applications as either benign or malicious. This is done by employing machine learning algorithms for the generation of behavior models of malicious and benign applications. A training phase, separate from the system, is the one which identifies the behavior of the applications. This module identifies Android applications into four classifications namely: Virus, Trojan, Spyware, Exploit or Benign.

For the training phase, behavior models for each type of Android application are generated by sampling a number of applications per each classification to run on different algorithms. Features of applications extracted from the previous module are translated into an .arff file format for Weka to be able to process the collected data. Currently, the module only generates the accuracy results of the chosen algorithms given feature sets from each type of malware and of the benign applications.

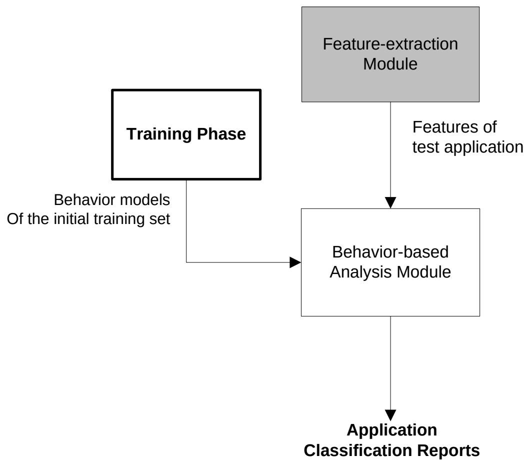
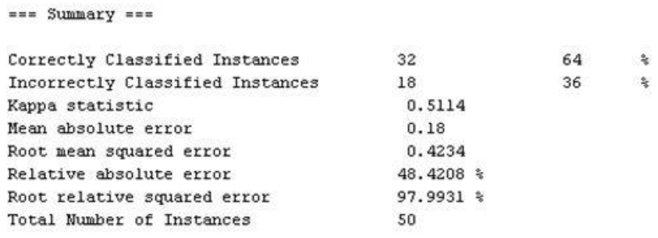

Feature-extraction Module
Training Phase
Features of test application
Behavior models Of the initial training set
Behavior-based Analysis Module
Application Classification Reports

Figure 3. Behavior-based Analysis Module

The algorithms used for this module include the Naïve Bayes algorithm for high bias in small data sets, the Decision Tree algorithms for its low bias 5 and the Logistic Regression algorithm to accommodate for adjustments in the features. 6 Based on studies which used a similar system setup for malware detection, the aforementioned algorithms performed best based on the garnered False Positive Ratings and True Positive Ratio from the tests. 7 8. The best performing algorithm based on percentage of correctly classified instances, Kappa statistic, precision, true positive rate and false positive rate.

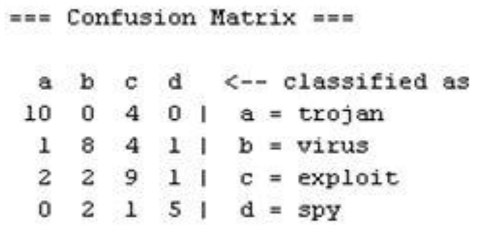


```
=== Summary ===

Correctly Classified Instances          32               64      %
Incorrectly Classified Instances        18               36      %
Kappa statistic                          0.5114
Mean absolute error                      0.18
Root mean squared error                  0.4234
Relative absolute error                 48.4208 %
Root relative squared error             97.9931 %
Total Number of Instances               50
```

Figure 4. Weka Statistics Result

In the statistics summary above (Fig. 4), the percentage of the *correct classified instances* is 64% and for the incorrect is 36%. The correctly and incorrectly classified instances, often called accuracy or sample accuracy, are the percentage of the test instances that were correctly and incorrectly classified. These also refer to the case where the instances are used as test data. When it comes to classification, correctly and incorrectly classified instances are the most important figures and will be used in the study. [17] With these figures, correctness of the classification of the applications to the different class labels can be determined. The numbers of applications and the classification are shown in the Confusion Matrix below, where a, b, c and d are the class labels which in the study's case, the malware types. There were 50 samples, so when you add up, a + b + c + d = 14 + 14 + 14 + 8.

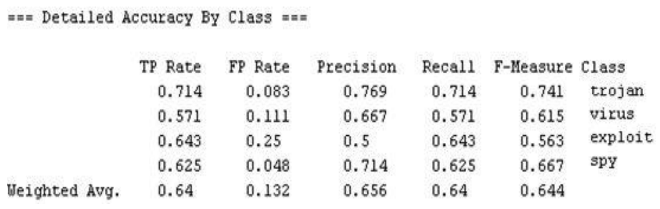


```
=== Confusion Matrix ===

  a  b  c  d   <-- classified as
 10  0  4  0 |  a = trojan
  1  8  4  1 |  b = virus
  2  2  9  1 |  c = exploit
  0  2  1  5 |  d = spy
```

Figure 5. Weka Confusion Matrix

Kappa statistic (see Fig. 4) measures the agreement of prediction between the true classes and the classifications. A value greater than 0.0 means that the classifier is doing better than the chance and a value of 1.0 signifies complete or perfect agreement. However, the error rates are used for numeric prediction rather than classification tasks which are not relevant in the study.



```
=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure Class
                 0.714     0.083     0.769     0.714     0.741    trojan
                 0.571     0.111     0.667     0.571     0.615    virus
                 0.643     0.25      0.5       0.643     0.563    exploit
                 0.625     0.048     0.714     0.625     0.667    spy
Weighted Avg.    0.64      0.132     0.656     0.64      0.644
```

Figure 6. Weka Detailed Accuracy Result

The *True Positive* (TP) rate is the proportion of applications which were truly classified to a certain class and how much part of the class was captured. It is also equal to the *Recall*. The percentage of Trojan-labeled applications that are classified as Trojans could be determined using TP. *False Positive* (FP) rate is the proportion of examples which were classified to a certain class but belongs to a different class. With FP, the percentage of the application classified as Trojans but are Virus-labeled can be generated. The *Precision* is the proportion of the examples which truly belong to a class among those where classified to a specific class. The F-Measure is simply *(2*Precision*Recall/(Precision + Recall)*, a combined measure for Recall and Precision. This could actually be interpreted as the weighted average of Precision and Recall. [19] ROC, on the other hand, is the measure of certainty of the algorithm with the classification made.

3 Machine Learning

D. *Naïve Bayes*

Naïve Bayes is the simplest form of Bayesian Network wherein given a class variable, all attributes are assumed to be independent. 9 The algorithm is able to classify by calculating the maximum likelihood of the attributes belonging to a certain class. Even with the interaction of certain attributes, the Naïve Bayes assumption does not lose predictive accuracy even if the actual probabilities are different. 10

An understanding of the Bayes classifier (1) is required to also understand the Naïve Bayes classifer. C is the class of an unobserved random variable to be learned. X denotes a feature vector variable while x denotes the value of the variable. Given the Bayes Classifer,

$$\boldsymbol{h^*(x) = \arg\max P\,(X = x|C = \mathrm{i})P(c = \mathrm{i})}$$

Equation 1. Bayes Classifier

which determines the *maxmim a posteriori probability* (MAP) given example x, proves difficult in providing direct estimation when there is high-dimensionality in feature space. This is because the Bayes classfier considers a *class-conditional probability distribution* (CPD) defined in $P(X = x|C = \mathrm{i})$ which relies on the dependence of each feature vector to another. Equation (2) describes a simplified assumption of the independence of features given the class. 15

$$\boldsymbol{f_i^{NB}(x) = \Pi_{j=1}^{n} P\left(X_j = x_j \middle| C = \mathrm{i}\right) P(C = \mathrm{i})}$$

Equation 2. Naive Bayes Classifier

E. *Decision Trees*

Decision Trees base the classification of instances by sorting feature vectors. In a decision tree, a node represents a feature to be classified and a branch represents the next possible value of a node. Decision trees may be interpreted as a set of rules for each path from the root to each leaf of the tree. 11 The rules employed by decision trees define how a split is created and how cases are classified as to what leaf is reached 13 These rules may also be derived from training data to be used for actual testing. 11

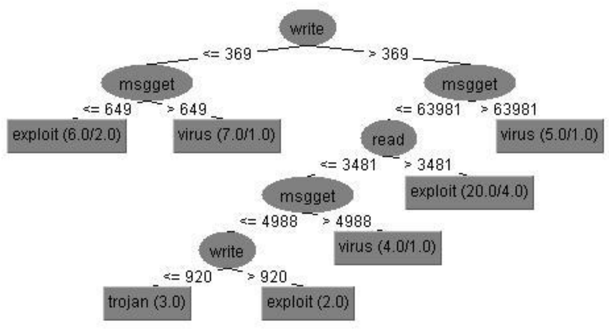


Figure 7. Decision Tree Sample

F. *J48*

J48 is an open source Java-based implementation of the C4.5 Decision Tree Algorithm. The algorithm splits the data set to build a certain node for a tree. The data with the highest information gain would be the one that most effectively splits the data set onto one class or another so this certain data is chosen. After choosing the data, a decision node is created to split based on the data chosen. The sublist obtained by splitting on the data with the highest information gain is the recursed and then added as children of the decision node. 11

G. *Random Forest*

Random Forest utilizes many classification trees to be able to classify an object based on the majority vote of classification generated by the trees. A tree is grown by first sampling a random number of *N* cases in the training set. For each input variable *M*, a number value *m* is used for each node to select randomly from the input variable to be used to split a node. After, the generated tree is fully grown as deep as possible. 13

H. *Multinomial Logistic Regression*

Since the study classifies more than two types of an Android application, Multinomial Logistic Regression (MLR) has to be utilized to produce polychotomous results over Logistic Regression (LR) which only produces a dichotomous result. In this note, MLR is an extension of LR which provides regression models by comparison of an arbitrary reference category to categories of an unordered response variable. Simply put, MLR utilizes multiple logistic regressions on a multi-category response variable that is unordered. Equation (3) illustrates a general multinomial logistic regression model where j is an identified variable and j' is the reference variable. X is an explanatory variable affects the resulting model. 14

$$log \frac{\Pr(Y = j)}{\Pr(Y = j')} = \alpha + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_k X_k$$

Equation 3. General Equation for MLR

4 Testing

The process of downloading applications, testing and choosing the algorithm, and classification of application are as follows:

a. *Learning Data Acquisition*

Known-malware applications with types of Trojan, Virus, Spyware and Exploit are gathered manually from the expert system VirusTotal which provides credibility for the learning data. VirusTotal uses an observed minimum of forty Anti-virus engines which scans the applications. Applications downloaded from VirusTotal are tagged as malware by at least ten anti-virus engines.

For the known-benign applications, they are gathered through the official Android Market using a mobile phone running on Gingerbread 2.3.3 Android OS. The applications are extracted from the mobile phone through the use of a File Manager application called Astro. Each application extracted produces an .apk file format of the application and these files are forwarded to VirusTotal to verify their status as benign applications.

Table 1. Count of Applications Downloaded

Type	Number of Apps Collected from VirusTotal	Number of Apps used for Training
Benign	135	50
Trojan	2334	50
Virus	257	50
Spyware	87	50
Exploit	184	50

The table shows an imbalanced distribution of applications downloaded for each classification from VirusTotal. When the applications are processed through the API, most applications are classified as Trojan, instead of the tagged classification.

b. *Application Acquisition*

A webcrawler is used to download applications from alternative markets. Some sites involve javascript download links which cannot be accessed by the crawler. Because of this, there are chosen markets where the webcrawler could work since they have the direct download links. Each market has a different webcrawler program to satisfy different settings.

Table 2. Count of Applications from Alt. Market

Alternative Android Market	Cell11	Appchina	Slideme
no. of applications downloaded	30	26	26

c. Application Virtualization

Applications are run inside an emulator to collect for logs of the behavior of these applications to be used by Weka.

A tool, Strace, is used to obtain the system calls made by the application that is running. These system calls that have been collected are then stored into a separate file along with its classification as benign or as one of the types of malware.

The systems calls to be collected are as follows: recv(), close(), brk(), open(), write(), msgget(), read(), lseek(), sigprocmask(), fork(), dup(), ioctl(), mprotect(), SYS_224.

d. Application Log Parsing

The application logs that have been generated by Strace are collected and injected into the parser program.

This parser program generates the ARFF (Attribute Relation File-Format) file to be used by Weka in classifying the applications. The program searches for specific system calls made by the application inside the log file. The count of these system calls are taken and then appended into the ARFF file. This is done for all desired system calls to be taken for all the application logs.

e. Algorithm Testing

The ARFF file generated by the parser program is fed into Weka for the classification of the applications. Different algorithms are tested to see whether which algorithm fares better.

The metrics to be checked for are the following: *1) True Positive Rate 2) Kappa Statistic 3) Receiver Operating Characteristic (ROC)*

Whilst the algorithms to be tested are the following: *1) J48 (J48graft) 2) Random Forest 3) Multinomial Logistic Regression 4) Naive Bayes.*

f. Application Log Parsing

The behavior logs of test applications are produced and then processed through the use of the parser for test applications. The parser works with a set dictionary of relevant features based from the features of the behavior model from the training phase. An ARFF file for each test application is produced by the parser which is used for comparison with the behavior-model of the most accurate algorithm.

The applications for this test are downloaded from different sources. These test applications came from VirusTotal and as well as form alternative Android markets namely: Slideme, Cell11 and Appchina.

g. Application Log Parsing

The behavior logs of test applications are produced and then processed through the use of the parser for test applications. The parser works with a set dictionary of relevant features based from the features of the behavior model from the training phase. An ARFF file for each test application is produced by the parser which is used for comparison with the behavior-model of the most accurate algorithm.

The applications for this test are downloaded from different sources. These test applications came from VirusTotal and as well as form alternative Android markets namely: Slideme, Cell11 and Appchina.

h. Application Log Parsing

AMDA application classification results stored in the database are compared to the classification report from VirusTotal. AMDA collates the results and uses a counting mechanism as to how many tags of applications are made by the AV engines as Trojan, Spyware, Exploit, Virus and Unclassified.

Again, the performance of the system is measured by the True Positive Rate, Kappa Statistic and the Receiver Operating Characteristic (ROC Curve).

5 ALGORITHM AND CLASSIFICATION RESULTS

The training phase for the system undergoes a rigorous process for being able to generate the best behavior model for the system. There is a total of 80 number of tests made. As mentioned in the earlier section, each algorithm is tested with three different feature selection methods and without a feature selection method used. This is done five times and for each test, the number of applications used varied in number.

Table 3. Number of Applications per Training Phase

Training Phase	Number of Applications
Test 1	50
Test 2	100
Test 3	150
Test 4	200
Test 5	250

For the training set, the Random Forest algorithm in Test 1 with Gain Ratio as the feature selection method and Test 5 with no feature selection method (See Figure. 8) garnered the best accuracy through measurement by True Positive Rate. Both tests achieved 78% accuracy. In addition, the Random Forest algorithm consistently outperformed the other algorithms. The behavior model from Training Phase Test 5 is chosen to be used for the system since it performed well even with a larger number of applications used and the test garnered a higher rate of ROC which means that the algorithm is definite with its classifications.

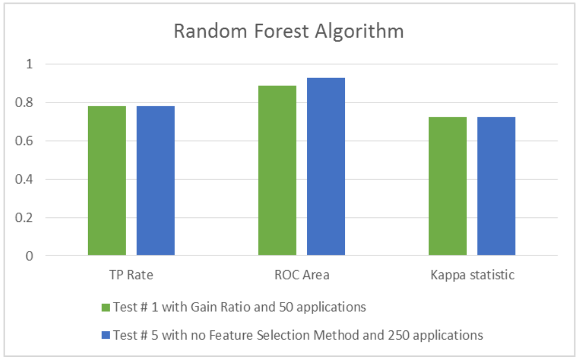


Figure 8. Summarized Results of Machine Learning Algorithms

After knowing the best algorithm for classification through the training phase, gathering and processing of test applications follows.

Table 4. Applications Classified by AMDA

Type of Android Application	Number of Applications Classified
Benign	35
Trojan	41
Spyware	54
Exploit	81
Virus	13

There are a total of 224 applications parsed by the system. The results are compared to the classification report from VirusTotal.

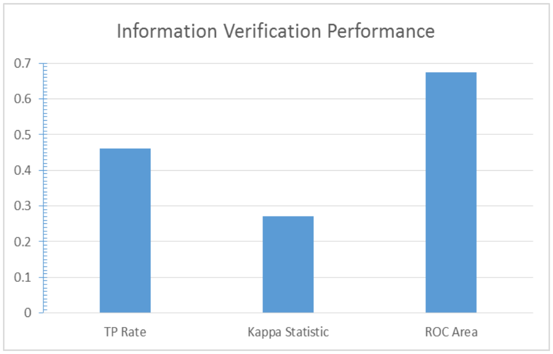


Figure 9. Information Verification Results

When the results of the AMDA System are validated to the results garnered through VirusTotal, TP Rate measurement exacted to 46.2%, Kappa Statistic to 27.17% and the ROC Area measured 67.5%. The results above constitute quite a low accuracy for classification of the types of Android applications. The ROC Area, being above the 50% mark, means that the system is mostly certain of its classifications. A low measure was garnered by the Kappa Statistic which means that the system encountered a dataset with mostly random attributes.

Further checking deep into the system calls is made to identify other measurement of analysis and problems. It is found that fourteen of the system call features exhibit the same characteristics for pairs of malware types. Virus and Exploit applications typically measure the same for these system calls. Trojan and Spyware applications are paired for the mentioned system calls.

With that information, it can be derived that malware applications exhibit the same behaviors which explains why the results of the classification is low. Instead of having 4 classifications for Malware, it is simplified into just Malware versus Benign classifications.

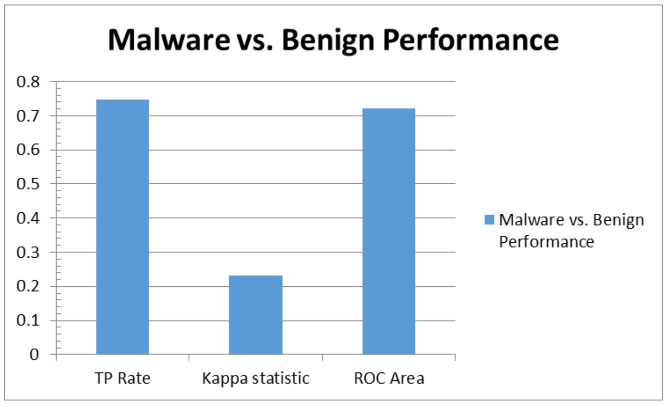


Figure 10. Malware vs. Benign Results

The TP Rate measurement increased to 74.7%, the Kappa Statistic to 23.17% and the ROC area as 72.1%. The TP Rate achieved a significantly higher value percentage compared to the previous result which indicates that the system is able to better correctly classify the applications. The Kappa Statistic measured is almost the same as the previous test. This is expected since the same dataset is used as with the previous test. The ROC Area still achieved a high measurement which indicates that the system is mostly certain of the classifications made.

6 Conclusion

The system, given the capability to classify unknown applications based from its data, can be used to categorize different Android applications in the market. With the web crawler at hand, the system has the potential to automatically download and classify new applications uploaded to the different alternative markets. Other than these, the system has the ability to classify malware to different types using behavior-based analysis. With this at hand, the system can act as an Anti-Virus that could easily provide classification results to users.

However, expert systems or different classification sources change classifications from time to time. This happens when more Anti-virus engines are able to classify applications as from when the application was first classified or because there are more and more malware families being identified. With this, there is a clear lack of standards in the classification scheme of applications. This lack of standards contributes to the futility of classifying malware into different classifications other than just classifying it as malware.

Another factor would be that malware families would have variants of other malware families which makes it even more difficult to distinguish between malware types [20].

7 Future Work

Further work to be done is the ability to detect advanced malware attacks such as Zero-day attack. Implementation of Behavior-based analysis with permission-based can also be done to determine malicious Android applications. Administrative User interface and an AMDA Android Application will allow easier analysis and access of the system.

References

1. T. Vennon, “Threat Analysis of the Android Market,” 2010. [Online]. Available:http://www.globalthreatcenter.com/wp-content/uploads/2010/06/Android-Market-Threat-Analysis-6-22-10-v1.pdf [Accessed: October 30, 2012]
2. K. Elish, D. Yao, and B. Ryder, “User-Centric Dependence Analysis For Identifying Malicious Mobile Apps,” in Proceedings of the IEEE CS Security and Privacy Workshop, 2012. San Francisco, CA.

3. T. Isohara, K. Takemori and A. Kubota, "Kernel-Based Behavior Analysis for Android Malware," in proceedings of the 2011 Seventh International Conference on Computational Intelligence and Security. Saitama, Japan. 2011.
4. I. Burguera., U. Zurutuza and S Tehrani, "Crowdroid: Behavior-Based Malware Detection System for Android," in Proceedings of the 18th ACM Conference on Computer and Communications Security, 2011. Chicago, IL. 17 October 2011.
5. T. Blasing and et al, "An Android Application Sandbox System for Suspicious Software Detection," in Proceedings of the 5th IEEE International Conference on Malicious and Unwanted Software, 2010.
6. Oracle, 2008. "Data Mining Concepts: Regression," 2008.[Online].Available:http://docs.oracle.com/cd/B28359_01/datamine.111/b28129/regress.htm#DMCON005 [Accessed: October 30, 2012]
7. B. Sans., "On the Automatic Categorisation of Android Applications," 2012. [Online]. Available: http://paginaspersonales.deusto.es/isantos/publications/2012/Sanz_2012_CCNC_Android_Apps_Categorisation.pdf. [Accessed: October 25, 2012]
8. A. Shabtai and C. Glezer, " "Andromaly" a behavioral malware detection framework for android devices," 2010. [Online]. Available: http://posgrado.escom.ipn.mx/biblioteca/%E2%80%9CAndromaly%E2%80%9D%20a%20behavioral%20malware%20detection.pdf
9. P. Flach and N. Lachiche, "Naïve Bayesian Classification of Structured Data," [Online]. Available: http://www.cs.bris.ac.uk/~flach/papers/mlj04-1BC-final2.pdf [Accessed: November 1, 2012]
10. H. Zhang, "The Optimality of Naïve Bayes," [Online]. Available: http://courses.ischool.berkeley.edu/i290-dm/s11/SECURE/Optimality_of_Naive_Bayes.pdf [Accessed: November 1, 2012]
11. S. Kotsiantis, I. D. Zaharakis and P. E. Pintelas, "Supervised Machine Learning: A Review of Classification and Combining Techniques," [Online]. Available: www.cs.bham.ac.uk/~pxt/IDA/class_rev.pdf [Accessed: November 2, 2012]
12. J. Chan, K. Chan, and A. Yeh, "Detecting the Nature of Change in an Urban Environment: A Comparison of Machine Learning Algorithms," American Society for Photogrammetry and Remote Sensing, , vol. 67, No. 2, pp. 213-225, February 2001.
13. L. Breiman and A. Cutler, "Random Forests," [Online]. Available: http://stat-www.berkeley.edu/users/breiman/RandomForests/cc_home.htm#intro [Accessed: November 3, 2012]
14. L. Moutinho and G.D. Hutcheson, "Dictionary of Quantitative Methods in Management," [Online]. Available: http://www.research-training.net/addedfiles/READING/MNLmodelChapter.pdf [Accessed: November 4, 2012]
15. I. Rish, "An Emprical Study of the naïve Bayes Classifer," [Online]. Available: www.cc.gatech.edu/~isbell/reading/papers/Rish.pdf [Accessed: November 5, 2012]
16. Weka, 2008, "Weka: Primer," 2012. [Online]. Available: http://weka.wikispaces.com/Primer [Accessed: November 5, 2012]
17. J. Tiedemann, "Interpreting Weka Output," [Online]. Available: http://www.let.rug.nl/tiedeman/ml06/InterpretingWekaOutput [Accessed: November 5, 2012]
18. J. He, "Linux System Call Quick Reference," [Online]. Available: http://www.digilife.be/quickreferences/qrc/linux%20system%20call%20quick%20reference.pdf [Accessed: November 7, 2012]
19. T.Borovicka, M.Jirina Jr., P. Kordik and M. Jirina, "Selecting Representative Data Sets," [Online]. Available: http://cdn.intechopen.com/pdfs/39037/InTech-Selecting_representative_data_sets.pdf [Accessed: December 2, 2012]
20. ESET Labs, 2013. "Trends for 2013: Astounding growth of mobile malware.," [Online]. Available: http://go.eset.com/us/resources/white-papers/Trends_for_2013_preview.pdf