Randomizing Smartphone Malware Profiles against Statistical Mining Techniques

Abhijith Shastry, Murat Kantarcioglu, Yan Zhou, and Bhavani Thuraisingham

Computer Science Department
University of Texas at Dallas
Richardson, TX 75080
abhijiths@utdallas.edu, muratk@utdallas.edu, yan.zhou2@utdallas.edu,
bxt043000@utdallas.edu

Abstract. The growing use of smartphones opens up new opportunities for malware activities such as eavesdropping on phone calls, reading e-mail and call-logs, and tracking callers' locations. Statistical data mining techniques have been shown to be applicable to detect smartphone malware. In this paper, we demonstrate that statistical mining techniques are prone to attacks that lead to random smartphone malware behavior. We show that with randomized profiles, statistical mining techniques can be easily foiled. Six in-house proof-of-concept malware programs are developed on the Android platform for this study. The malware programs are designed to perform privacy intrusion, information theft, and denial of service attacks. By simulating and tuning the frequency and interval of attacks, we aim to answer the following questions: 1) Can statistical mining algorithms detect smartphone malware by monitoring the statistics of smartphone usage? 2) Are data mining algorithms robust against malware with random profiles? 3) Can simple consolidation of random profiles over a fixed time frame prepare a higher quality data source for existing algorithms?

1 Introduction

Compared to conventional mobile phones, smartphones are built to support more advanced computing needs modern custom software demands. An unpleasant byproduct of the ongoing smartphone revolution is its invitation to malicious exploits. As smartphone software grows more complex, more malware programs will be created to attempt to exploit specific weaknesses in smartphone software [4, 6]. Smartphones of end users all together constitute a large portion of the powerful mobile network. Having access to the enormous amount of personal information on this network is a great incentive for the adversary to attack the smartphone mobile world.

Malicious activities on mobile phones are often carried out through lightweight applications, scrupulously avoiding detection while leaving little trace for malware analysis. Over the years many malware detection techniques have been proposed. These techniques can be roughly divided into two groups: static analysis and dynamic analysis. Static analysis techniques discover implications of

unusual program activities directly from the source code. Although static analysis is a critical component in program analysis, its ability to cope with highly dynamic malware is unsatisfactory. A number of obfuscation techniques have been shown to easily foil techniques that rely solely on static analysis [14]. Dynamic analysis (also known as behavioral analysis) identifies security holes by executing a program and closely monitoring its activities [24]. Information such as system calls, network access, and files and memory modifications is collected from the operating system at runtime [18]. Since the actual behavior of a program is monitored, threats from dynamic tactics such as obfuscation are not as severe in dynamic analysis. However, dynamic analysis can not guarantee a malicious payload is always activated every time the host program is executed.

We follow a similar perspective of dynamic analysis by analyzing real-time collections of statistics of smartphone usage. Metrics of real-time usage are recorded for analysis. We choose the Android platform in our study. Android is open source and apparently has a solid customer base given that many devices are using this platform. For the convenience of security analysis on this platform, we developed custom parameterized malware programs on the Android platform. These malware programs can target the victim for the purpose of denial of service attacks, information stealing, and privacy intrusion. Our second contribution is the empirical analysis of the weaknesses of data mining techniques against mobile malware. We demonstrate that a malware program with unpredictable attacking strategies is more resilient to commonly used data mining techniques.

The rest of the paper is organized as follows. Section 2 presents the related work in the field of malware detection. Malware detection techniques developed for general-purpose use and those designed specifically for mobile phones are discussed in this section. Section 3 presents six malware programs and their tuning parameters. Section 4 discusses the experimental setup and the data collected for analysis. Experimental results are also presented. Conclusions are presented in Section 5.

2 Related Work

We first give a broad overview of malware detection techniques in general since those techniques share common roots with techniques specifically developed for smartphone malware. In the second part of this section, we discuss work directly related to mobile malware detection.

2.1 Malware Detection Technique

Techniques used for malware detection can be categorized broadly into two categories: anomaly-based detection and signature-based detection. Anomaly-based detection techniques use the knowledge of what constitutes normal behavior to decide the maliciousness of a program. For example, a rule-based system decides whether a program is benign or malicious based on a pre-defined set of rules.

Signature-based detection, on the other hand, makes use of static characterization of known malicious software [24]. Detection techniques generally follow three different approaches: static, dynamic, or hybrid analysis. A static approach typically attempts to detect malware without executing the program under inspection, while a dynamic approach attempts to detect malicious behavior during program execution or after program execution. Thus the dynamic approach is often referred to as behavior based analysis. Hybrid techniques leverage the advantages of the previous two approaches by combining them as described in [17].

Lee and Stolfo [11] propose to use association rules and frequent episodes in intrusion detection systems. The association rules and frequent episodes can be collectively referred to as a rule set. Rule sets are created for various security-critical aspects of the target host. These rule sets serve as the knowledge of what activities are considered as normal on the host.

Hofmeyr et al. [8] propose a technique that monitors system call sequences in order to detect maliciousness. Initially profiles representing the normal behavior of a system are developed. The behavior is characterized by sequences of system calls. Hamming distance is used to determine how closely a system call sequence resembles another. Thresholds are used to determine whether a process is anomalous. Okazaki et al. [15] also propose a detection method based on the frequency of system calls.

Static anomaly-based detection techniques use the characteristics of the file structure of the program to identify malicious code. A major advantage of static anomaly-based detection is that it is possible to detect malware without having to execute the program containing the malware. Stolfo et al. [21] describe fileprint (n-gram) analysis as a means for detecting malware. Many other existing anomaly-based malware detection mechanisms use a hybrid approach [17].

2.2 Malware Detection in Mobile Phones

Malware detection techniques developed for use on the computer platform cannot be directly used in a mobile environment due to limited resources and processing capabilities of a mobile phone. Many anomaly-based and signature-based detection techniques, mostly using a dynamic approach, have been proposed to detect malware on mobile phones.

Zhou et al. [23] present permission-based behavioral footprinting and heuristic-based filtering techniques for identifying both known and unknown malware in the Android family. They first filter out Android apps based on the permissions required to grant wrongdoings on the phone, and then define suspicious behaviors of malicious apps and use them to detect zero-day malware.

Yap and Ewe [22] propose a behavior checker solution that detects malicious activities in a mobile system. A proof of concept scenario using a Nokia mobile phone on the Symbian operating system is provided. Bose et al. [1] propose a behavioral detection framework that employs a temporal logic approach to detect malicious activities over time. An efficient representation of malware behaviors is proposed based on a key observation that the logical ordering of an application's

actions over time often reveals malicious intent even when each action alone may appear harmless.

Kim et al. propose a detection mechanism based on power signatures [10]. The technique can detect and analyze previously unknown energy depletion threads based on a collection of power signatures. Moreau et al. [13] use artificial Neural Networks (ANNs) to detect anomalous behavior indicating a fraudulent use of the operator services. An example of such behavior is unusually high call rate. Cheng et al. [3] propose SmartSiren, a virus detection and alert system for smartphones. SmartSiren was evaluated by detecting SMS viruses by monitoring the amount of SMSs sent by a single device.

Schmidt et al. [19] extract features representing device state from a smartphone running the Symbian OS. These extracted features are used for anomaly detection to distinguish between normal and abnormal behavior. The processing of the extracted features was performed on a remote server. Dixon and Mishra [5] propose a rootkit and malware detection mechanism for smartphones in which processing is performed on a computer which is connected to the mobile device. An implementation on the Android platform is also provided.

Shabtai et al. propose a behavioral malware detection framework for android devices [20]. The framework includes a host-based malware detection system that continuously monitors various features and events obtained from the mobile devices, and then applies machine learning anomaly detectors to classify the collected data as benign or malicious. They develop four malicious applications on the Android platform and evaluate the proposed framework. They show that such a behavioral malware detection scheme is able to detect unknown malware programs.

3 MALWARE SETUP

We developed six different parameterized malware programs on the Android platform. These malware programs perform privacy intrusion, information theft attacks, and denial of service attacks. By varying the parameters of the malware programs, different profiles of the same malware can be generated. Moreover, the parameters themselves can be randomized (with an expected mean value) rather than being a fixed value. By randomizing the parameters, interesting malware profiles can be prepared for further analysis.

We assume that either through a direct installation or an indirect installation (through the payload of a benign application), the victim's mobile phone is infected with the developed malware. All malware programs were developed and tested on a Samsung Captivate smartphone running on the Android platform. One important thing to know about the Android framework is that applications run in sand boxes (virtual machines), and therefore do not impact other applications in the system. Moreover, all permissions required by the application running on the Android platform (such as Internet access, microphone access) have to be declared, which is prompted to the user when the application is installed. Again the assumption we are leaning on allows us to get away from any

practical difficulties of installing the developed malware programs in a furtive manner.

3.1 Call Recorder

The *Call Recorder* malware performs eavesdropping on incoming and outgoing phone calls. Both incoming and outgoing calls are recorded. The recorded file is kept locally on the phone. A configuration option is provided to upload the recorded file to a server. This malware attempts to compromise the privacy of the person using the infected mobile phone. Parameters of this malware include:

- MAX_DURATION—maximum duration a phone call is recorded
- MAX_FILESIZE—maximum size of the recorded file
- NUM_SKIPPED_CALLS—specifying that only every $(n+1)^{th}$ phone call is recorded, where n is the value of NUM_SKIPPED_CALLS.
- INTERVAL_RECORD—specifying the length of every recording (after each sleep) during a phone call
- INTERVAL_SLEEP—specifying the duration in which the malware sleeps (stops recording)
- SHOULD_UPLOAD—uploading the recorded content to a server
- DELETE_LOCAL—deleting the local copy of the recording output

3.2 DoS Malware

Dos Malware performs a Denial of Service (DoS) attack. Upon loading this application, it spawns many threads. Each thread performs a large number of multiplications between two randomly generated numbers. As the number of threads increases, the phone starts becoming unresponsive. When the number of threads spawned is above 200, the phone hangs and has to be rebooted. Thus, this malware paralyzes the device by driving the CPU beyond its limit. Parameters of this malware are:

- MAX_THREADS—number of threads spawned by the malware
- NUM_MULTIPLICATIONS—number of multiplications performed by each thread
- INTERVAL_RESTART—duration after which all the spawned threads are killed before new ones are spawned
- INTERVAL_SLEEP—duration in which the malware sleeps before new threads are spawned

3.3 Mass Uploader

As the name suggests, this malicious application uploads the contents of the memory device of the mobile phone to a server. Thus, it is designed to steal information from the device. Other than uploading, this malware can also download content from a server. When this application is started, it begins the process of uploading and downloading content to/from a server. Parameters of this malware are:

– UPLOAD/DOWNLOAD_BW—the upload/download bandwidth limits for the malicious application
– UPLOAD/DOWNLOAD_INTERVAL—the duration after which an upload/download is performed by the malicious application
– UPLOAD/DOWNLOAD_INTER_LIM—specifying the limit of the amount of data sent (burst) in one upload/download attempt

Note that memory private to an application is protected by linux permissions on Android. Therefore normally other service cannot access it. We assume a root exploit has enabled the application to elevate to root and steal sensitive data.

3.4 Smart Recorder

Smart Recorder performs eavesdropping on incoming and outgoing phone calls from specific phone numbers. These specific phone numbers are read from a server whenever a phone call is made. The specific phone numbers can be changed at run time. After recording a phone call, the recorded file is uploaded to a server. This malware gives the attacker more control over the recorded phone conversations. Specific phone calls can be targeted as the attacker tries to compromise the privacy of the person using the infected mobile phone. Parameters of this malware are:

– MAX_DURATION—maximum duration that the phone call is recorded.
– MAX_FILESIZE—maximum size of the recorded file
– INTERVAL_RECORD—length of every recording (after each sleep) during a phone call
– INTERVAL_SLEEP—duration in which the malware sleeps (stops recording)

3.5 Spy Camera

Spy Camera can spy on the unsuspecting user by taking snap shots from the mobile phone camera every few seconds. These snapshots can be uploaded to a server. Thus, this malware compromises the privacy of the user. When the malware takes a snap from the mobile phone, the user is not notified in any way (by sound or other notifications) that a picture has been taken from the mobile phone camera. Parameters of this malware are:

– SNAP_INTERVAL—duration after which a snap is taken from the camera on board the mobile device and stored locally on the phone
– PIC_DSAMPLE_RATIO—specifying the down sample ratio that impacts the quality of the pictures taken form the camera
– PIC_COMP_QUALITY—specifying the amount of compression the raw image is subjected to before saving the image
– SHOULD_UPLOAD—uploading the picture to a server
– DELETE_LOCAL—deleting the local copy of the pictures taken

3.6 Spy Recorder

Spy Recorder can remotely turn on the microphone of the mobile phone and start recording any voice input. The recording is initiated and terminated by a phone call from a specific number that a spy has registered. The microphone is turned on when a phone call from a specific number is made to the victim's phone. The call is automatically rejected afterwards. Similarly, making another phone call from a specific number will turn off the microphone. The entire process is completed without the user's attention. This spyware can be used to record conversations duration important meetings. The recorded file can be stored on the mobile device for later retrieval or can be uploaded to a server. Parameters of this malware are:

- MAX_DURATION—maximum duration that the conversation is recorded
- MAX_FILESIZE—maximum size of the recorded file
- INTERVAL_RECORD—length of every recording (after each sleep) during a phone call
- INTERVAL_SLEEP—duration in which the malware sleeps (stops recording)
- SHOULD_UPLOAD—uploading recorded conversation to a server
- DELETE_LOCAL—deleting the local copy of the recorded conversation

4 EXPERIMENTAL ANALYSIS

We now discuss the experiments in which we investigate the weaknesses of common data mining detection techniques. We present metrics characterizing the behavior of an application and the data sets used in the experiments data. We also discuss the data mining tools used in our experiments and the evaluation metrics.

4.1 Run-time Behavior Metrics

The run-time behavior of a program can be defined using the statistics of usage of a smartphone. In our experiments, we record the run-time behavior of an application as a set of pre-defined features while the application is running. The collected feature sets are the data source for the data mining techniques later.

The features used in this study characterize the typical behavior of an application. They include various metrics, such as CPU consumption, network traffic, memory usage, and battery (power) consumption. Table 1 lists all the features that need to be recorded in real-time. A light-weight utility program was developed for collecting these features every five seconds and storing them in a database on the mobile phone. This application runs as a service in the background.

Whenever an application is running, the feature extraction utility collects the features from the running application. Once the features are collected, the next step involves training a classifier on the collected feature vectors. After the classifier is trained, dynamic decisions can be made for a running application by

Table 1. List of features extracted.

Feature Category	Feature
CPU	cpu_totutil, cpu_totproccount, cpu_userproccount cpu_load_avg1min, cpu_load_avg5min, cpu_load_avg15min
Memory	mem_tot, mem_free, mem_buffer mem_cached, mem_active, mem_inactive mem_dirty, mem_writeback, mem_pageanon mem_mapped, mem_anon, mem_file
Battery	btr_lvl_rem, btr_status, btr_temp btr_volt, btr_lvl_change
Network	net_cell_upd_tx_pkts, net_cell_upd_tx_bytes net_cell_upd_rx_pkts, net_cell_upd_rx_bytes net_wifi_upd_tx_pkts, net_wifi_upd_tx_bytes net_wifi_upd_rx_pkts, net_wifi_upd_rx_bytes

the classifier using the new collection of the features of the running application. Proper actions can be performed after detection, such as notifying the user when the classifier flags an application as malicious.

4.2 Data Sets

We developed a feature extractor application for collecting data. It runs as a background service on the Android phone. Features of the benign and malicious applications are collected from the Android OS while they are running. Using the feature extractor application, we create data sets from 20 benign applications and 6 malware programs over a period of 10 minutes, resulting 120 feature vectors per application. 10 of the benign applications were tools and the other 10 were games. The applications used in our experiments are listed in Table 2.

Table 2. Applications used in the experiments

Malicious	Benign (tools)	Benign (games)
Call Recorder	MusicPlayer	AngryBirds
DoS Malware	Phone	AirControlLite
Mass Uploader	Browser	SmartTacToe
Smart Recorder	Calculator	Snake
Spy Camera	Youtube	Minesweeper
Spy Recorder	Calendar	3DBowling
	Clock	Solitaire
	Contacts	RacingMoto
	Market	DragonFly
	Memo	DragRacing

Features from malware programs were collected when no limits (such as bandwidth, CPU usage) were imposed on the malware programs. This malware profile

is referred to as the general profile. In addition, features of malware programs with randomized profiles were also collected. For each malware program, five different randomized profiles were created. Thus the entire data set consists of feature vectors from 20 benign applications, 6 malware programs, and 30 randomized profiles of malware programs, in the total of 56 applications. Each of the randomized profile varies from the other with respect to the amount of randomization, i.e., the amount of deviation from the mean value of a parameter. For example, Rand$x$ refers to a profile in which the variance from the mean value of the parameters is x.

4.3 Data Mining Tools

Five data mining algorithms in WEKA [7] are selected to build the classifiers in our study. The algorithms are: decision tree (DT), logistic regression (LR), naïve Bayes (NB), artificial neural network (ANN), and support vector machine (SVM). Classifiers typically operate in two phases: training and testing. During the training phase, a classifier is trained on input vectors with proper class values. The classifier then builds a model that generalizes on data in the entire domain. After the training phase, the classifier can be used to make predictions for unseen instances, and it is known as the test phase.

Decision trees are tree-structured predictive models used in many application domains of Machine Learning. Many different types of decision trees exist. In our study we built a decision tree model using the C4.5 algorithm [16].

Logistic Regression is a type of predictive model that can be used when the target variable is a categorical variable with two categories and thus is suitable for our study. During the training phase, the logistic regression algorithm builds a model that is similar to fitting a polynomial to a set of data values. This model is then used to predict class labels during the test phase.

A naïve Bayes classifier is a probabilistic classifier based on the Bayes theorem with strong (naïve) independence assumptions. During the training phase, a model is built which is described by a set of probabilities. An important limitation of this classifier is that input features must all be discretized. It cannot directly handle continuous values. Continuous valued features can be handled using a mixture of Gaussians [12].

A neural network classifier builds a graph model that consists of a set of inter-connecting artificial neurons in its training phase. The neural network [12] exploits patterns in data by modeling complex relationships between the input and the target output. Typically, training neural network models takes more time than that for other models.

A standard support vector machine (SVM) algorithm builds a predictive model by constructing a high-dimensional hyper-plane that discriminates between two categories of data. Given a set of training examples, a hyper-plane that maximizes the distance to the closest training examples on either side is chosen as the optimal separating hyper-plane. The SVM methods have demonstrated great success in many application domains since it was first introduced to the machine learning research community [2].

4.4 Evaluation

The following standard metrics were used to measure the performance of the selected data mining algorithms:

1. True Positive Rate (TPR): Proportion of positive instances (feature vectors of malicious applications) classified correctly.
2. False Positive Rate (FPR): Proportion of negative instances (feature vectors of benign applications) misclassified.
3. Total Accuracy: Proportion of absolutely correctly classified instances, either positive or negative.
4. Receiver Operating Characteristic (ROC) - Area Under Curve (AUC): The ROC curve is a graphical representation of the trade-off between the TPR and FPR for every possible detection cut-off. AUC is the area under this curve.

4.5 Experiments

We now describe in detail two separate experiments we have performed on the datasets of the benign and malicious applications. We also discuss the experimental results and their implications.

Experiment 1 The first experiment evaluates the performance of the five data mining techniques when the adversary does not spread out the attacks in an unpredictable manner. Table 3 shows the 10-fold cross validation results when the data set consisted of the 20 benign applications and 6 malware programs. Four out of the five algorithms work fairly well with a classification accuracy of more than 95%. Naïve Bayes turned out to be a disappointing exception. Violation of the independence assumption in the data may be the main reason that hampers the performance of the naïve Bayes classifier. This experiment implies that statistical analysis in these data mining algorithms is in general applicable when an attack is not disseminated.

Table 3. 10-fold cross validation results on the 20 benign applications and 6 malware programs.

Classifier	Accuracy	TPR	FPR	AUC
DT	99.615	0.988	0.001	0.992
LR	99.231	0.988	0.006	0.995
NB	82.692	0.303	0.016	0.788
ANN	99.808	0.996	0.001	0.999
SVM	95.416	0.832	0.009	0.911

Table 4 and Table 5 present the cross validation results when the data set consists of 10 out of the 20 genuine applications and the 6 malware programs.

The genuine applications chosen in Table 4 and Table 5 are tools and games, respectively. All classifiers except naïve Bayes perform well with respect to the cross validation results. When games are used in the data sets instead of tools, the classification accuracy is slightly better for four classifiers. This result is consistent with the observation made in a similar experiment in [20].

Table 4. 10-fold cross validation results on 10 benign applications (tools) and 6 malware programs.

Classifier	Accuracy	TPR	FPR	AUC
DT	99.531	0.993	0.003	0.996
LR	99.167	0.989	0.007	0.996
NB	71.563	0.292	0.030	0.849
ANN	99.792	0.997	0.002	1.000
SVM	98.698	0.986	0.0125	0.987

Table 5. 10-fold cross validation results on 10 benign applications (games) and 6 malware programs.

Classifier	Accuracy	TPR	FPR	AUC
DT	99.948	0.999	0.000	0.999
LR	99.271	0.994	0.008	0.995
NB	74.688	0.339	0.008	0.829
ANN	99.948	1.000	0.000	1.000
SVM	93.490	0.840	0.008	0.916

Experiment 2 This experiment demonstrates the performance of the five data mining algorithms on a non-randomized malware profile (marked as General in the following plots) and five randomized malware profiles, namely Rand0, Rand5, Rand25, Rand50, and Rand75. Each randomized profile is generated by varying the parameters of each malware program as described earlier. Through this experiment, we hope to answer the two questions we raised earlier:

- Are data mining algorithms robust against malware with random attacking activities?
- Can simple consolidation of activities over a fixed time frame prepare a higher quality data source for existing algorithms?

To answer the second question, we experimented on datasets in which instances are consolidated by sequentially averaging over n consecutive samples we have extracted in *Experiment 1*, where n =1, 5, 10, and 50, marked as 5sec, 25sec, 50sec, and 250sec interval respectively in the figures. Figures 1 illustrates the

average classification accuracy over all five classifiers on 6x6 training-test data pairs. The training data consists of all the genuine applications and the malware programs with each of the six profiles shown as a label on the x-axis. Each cluster in a plot shows the average classification accuracy on six test sets—the same datasets of six different profiles {*General, Rand0, Rand5, Rand25, Rand50, Rand75*}, labeled with specific bar patterns as shown in the figures.

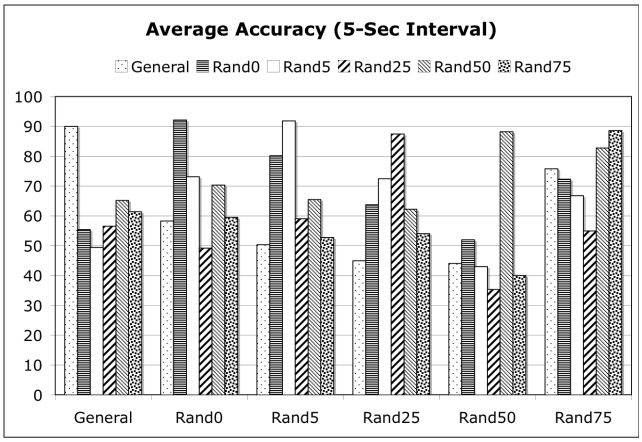

Fig. 1. Average Accuracy over all algorithms on 5sec data profiles.

Due to space limitations, we do not show the figures of the classification accuracy of each individual algorithm. In general, none of the classifiers was able to consistently outperform the others as the training and test data varies according to different random profile configurations. The decision tree algorithm performs best on datasets of all six profiles when the training data is extracted from the Rand75 profile and instances are formed by averaging 50 consecutive samples in each 250-second interval. Logistic regression performs best when the training data comes from the Rand25 profile and all instances in each data set is formed by averaging 10 consecutive samples in each 50-second interval. Naïve Bayes performs best with the training data of the Rand50 profile and instances are formed by averaging 50 consecutive samples. ANN performs best with training data of the Rand75 profile and samples of every 5-second interval. SVM performs best with training data of the Rand75 profile and instances are formed by averaging 50 consecutive samples. As can be seen, behavioral analysis may become very difficult when malware exhibits random behavior. Figure 2 shows the classification accuracy of each algorithm averaged over the 6x6 profile pairs.

Other key observations are: 1.) When the training set consists of benign applications and general malware programs while the test set consists of randomized profiles of malware programs, the classification accuracy is very poor

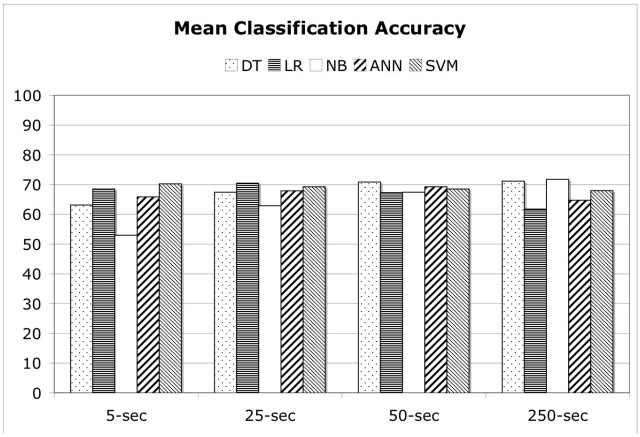

Fig. 2. Mean Accuracy of each algorithm on all data profiles.

irrespective of the classifiers. In most cases the classification accuracy is below 70%. This has important implications that a behavioral analysis-based malware detection scheme will fail when the training set consists of just general malware programs; 2.) Another observation is that when training set includes a randomized malware profile say, Rand-x and tests are carried out on another randomized malware profile, say Rand-y, classification accuracy is good when x is close to y, in general. For instance, using the decision tree classifier, training on the Rand50 profile and testing on the Rand75 profile gives a classification accuracy of around 86%. Some anomalies exist to this trend such as training on a Rand75 profile using a decision tree classifier.

Figures 3— 5 illustrate performance change as instances are formed by averaging samples in longer durations. As can be observed in Figure 3, the majority of performance change is positive when we average the samples every 25 seconds. This implies the mean point of a few consecutive samples serves better as an instance in the data set. Further consolidation using longer durations of 50 seconds and 250 seconds do not appear to improve the performance further except for the Rand50 profile. For individual algorithms, we observe significant performance improvement from the *decision tree* and *naïve Bayes* classifiers. The *Logistic Regression*, *Artificial Neural Network*, and *Support Vector Machine* classifiers all experienced an initial increase followed by slight decreases in classification accuracy as instances are formed over a longer duration. SVM is the most consistent one among the five classifiers. In general, sample consolidation does seem to improve classifier performance.

5 CONCLUSIONS

We developed six custom parameterized malware programs on the Android platform. These malware programs can perform a variety of malicious activities on

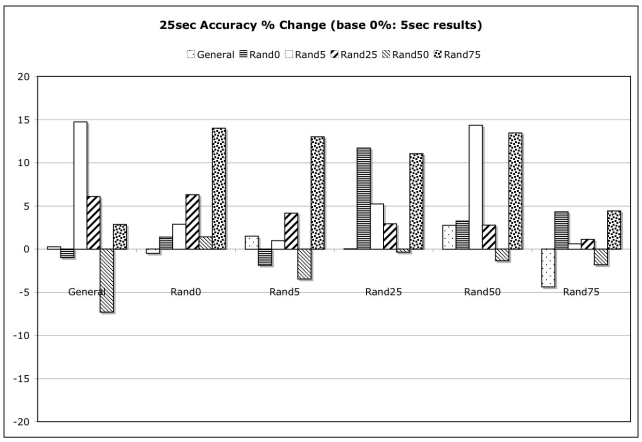

Fig. 3. Average Accuracy% change over all algorithms on 25sec data profiles.

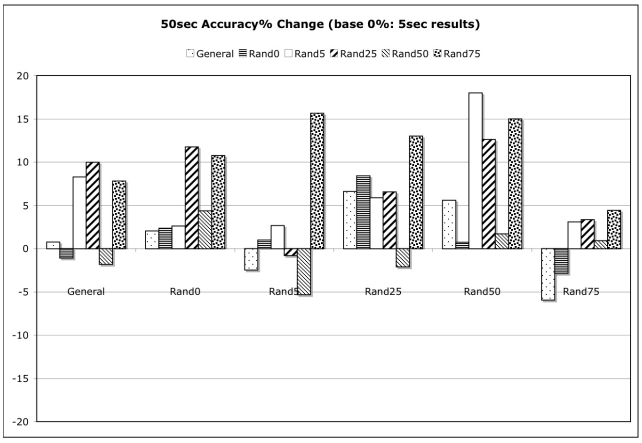

Fig. 4. Average Accuracy% over all algorithms on 50sec data profiles.

the victim's smartphone. We demonstrate that, although a data mining algorithm can be very successful when the training and the test data follow similar distributions, its performance is unsatisfactory on randomized profiles of the same malware programs. Therefore it is necessary to search for solutions that can better handle random behavioral patterns of malware programs. We also demonstrate that simple consolidation may effectively improve classification performance. In the future, we plan to expand the datasets by developing additional malware applications and including real-world malware, and moreover, search for reliable ways to improve detection in a volatile environment using adversarial classification techniques [9]. We also plan to compare data mining techniques with existing practical techniques for malware detection such as permission-based filtering and behavioral footprint matching methods [23].

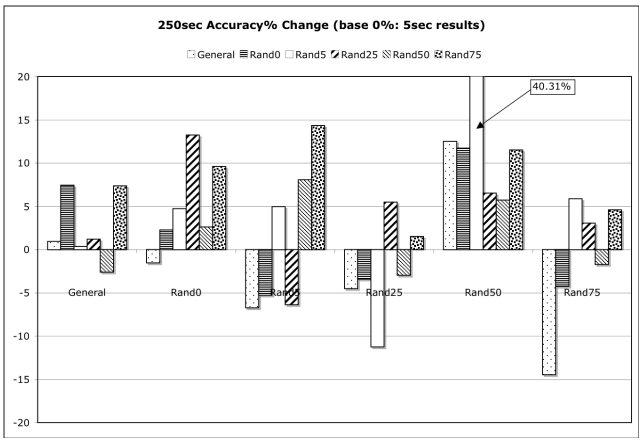

Fig. 5. Average Accuracy% over all algorithms on 250sec data profiles.

6 Acknowledgments

This work was partially supported by Air Force Office of Scientific Research MURI Grant FA9550-08-1-0265, National Institutes of Health Grant 1R01LM009989, National Science Foundation (NSF) Grant Career-CNS-0845803, and NSF Grants CNS-0964350, CNS-1016343.

References

1. Bose, A., Hu, X., Shin, K.G., Park, T.: Behavioral detection of malware on mobile handsets. In: Proceeding of the 6th international conference on Mobile systems, applications, and services. pp. 225–238. MobiSys '08, ACM, New York, NY, USA (2008)
2. Boser, B.E., Guyon, I.M., Vapnik, V.N.: A training algorithm for optimal margin classifiers. In: Proceedings of the 5th Annual ACM Workshop on Computational Learning Theory. pp. 144–152. ACM Press (1992)
3. Cheng, J., Wong, S.H., Yang, H., Lu, S.: Smartsiren: virus detection and alert for smartphones. In: Proceedings of the 5th international conference on Mobile systems, applications and services. pp. 258–271. MobiSys '07, ACM, New York, NY, USA (2007)
4. Christodorescu, M., Jhacomputer, S.: Testing malware detectors. In: In Proceedings of the 2004 ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA 2004. pp. 34–44. ACM Press (2004)
5. Dixon, B., Mishra, S.: On rootkit and malware detection in smartphones. In: Dependable Systems and Networks Workshops (DSN-W), 2010 International Conference on. pp. 162 –163 (28 2010-july 1 2010)
6. Gary McGraw, G.M.: Attacking malicious code: a report to the infosec research council. IEEE Software pp. 33–41 (2000), magazine article
7. Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, I.H.: The weka data mining software: an update. SIGKDD Explor. Newsl. 11, 10–18 (November 2009)

8. Hofmeyr, S.A., Forrest, S., Somayaji, A.: Intrusion detection using sequences of system calls. J. Comput. Secur. 6, 151–180 (August 1998)
9. Kantarcioglu, M., Xi, B., Clifton, C.: Classifier evaluation and attribute selection against active adversaries. Data Min. Knowl. Discov. 22, 291–335 (January 2011)
10. Kim, H., Smith, J., Shin, K.G.: Detecting energy-greedy anomalies and mobile malware variants. In: Proceeding of the 6th international conference on Mobile systems, applications, and services. pp. 239–252. MobiSys '08, ACM, New York, NY, USA (2008)
11. Lee, W., Stolfo, S.J.: Data mining approaches for intrusion detection. In: Proceedings of the 7th conference on USENIX Security Symposium - Volume 7. pp. 6–6. USENIX Association, Berkeley, CA, USA (1998)
12. Mitchell, T.M.: Machine Learning. McGraw-Hill, New York (1997)
13. Moreau, Y., Shawe-taylor, P.B.J., Stoermann, C., Ag, S., Vodafone, C.C.: Novel techniques for fraud detection in mobile telecommunication networks. In: ACTS mobile summit (1997)
14. Moser, A., Kruegel, C., Kirda, E.: Limits of static analysis for malware detection. In: Twenty-Third Annual Computer Security Applications Conference (ACSAC 2007. pp. 421–430 (2007)
15. Okazaki, Y., Sato, I., Goto, S.: A new intrusion detection method based on process profiling. In: Applications and the Internet, 2002. (SAINT 2002). Proceedings. 2002 Symposium on. pp. 82 –90 (2002)
16. Quinlan, J.R.: C4.5: programs for machine learning. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (1993)
17. Rabek, J.C., Khazan, R.I., Lewandowski, S.M., Cunningham, R.K.: Detection of injected, dynamically generated, and obfuscated malicious code. In: Proceedings of the 2003 ACM workshop on Rapid malcode. pp. 76–82. WORM '03, ACM, New York, NY, USA (2003)
18. Rieck, K., Holz, T., Willems, C., Dssel, P., Laskov, P.: Learning and classification of malware behavior. In: Zamboni, D. (ed.) Detection of Intrusions and Malware, and Vulnerability Assessment, Lecture Notes in Computer Science, vol. 5137, pp. 108–125. Springer Berlin / Heidelberg (2008)
19. Schmidt, A., Schmidt, H., Clausen, J., Camtepe, A., Albayrak, S.: Enhancing security of linux-based android devices. Image Rochester NY (2008)
20. Shabtai, A., Kanonov, U., Elovici, Y., Glezer, C., Weiss, Y.: "andromaly": a behavioral malware detection framework for android devices. Journal of Intelligent Information Systems pp. 1–30 (2011)
21. Stolfo, S.J., Wang, K., jen Li, W.: Worms 2005 columbia ids lab fileprint analysis for malware detection 1. In: 6th IEEE Information Assurance Workshop (2005)
22. Yap, T.S., Ewe, H.T.: A mobile phone malicious software detection model with behavior checker. In: Human.Society@Internet. pp. 57–65. Lecture Notes in Computer Science (2005)
23. Zhou, Y., Wang, Z., Zhou, W., Jiang, X.: Hey, you, get off of my market: Detecting malicious apps in official and alternative android markets. In: In Proceedings of the 19th Network and Distributed System Security Symposium (NDSS 2012) (2012)
24. Zolkipli, M.F., Jantan, A.: Malware behavior analysis: Learning and understanding current malware threats. Network Applications, Protocols and Services, International Conference on 0, 218–221 (2010)