Malware Analysis on Android Using Supervised Machine Learning Techniques

Md. Shohel Rana*, Andrew H. Sung
The University of Southern Mississippi, School of Computing, Hattiesburg, MS 39406, USA.

* Corresponding author. Tel.: +1 (929) 331-7300; email: md.rana@usm.edu
Manuscript submitted June 27, 2018; accepted September 5, 2018.
doi: 10.17706/ijcce.2018.7.4.178-188

Abstract: In recent years, a widespread research is conducted with the growth of malware resulted in the domain of malware analysis and detection in Android devices. Android, a mobile-based operating system currently having more than one billion active users with a high market impact that have inspired the expansion of malware by cyber criminals. Android implements a different architecture and security controls to solve the problems caused by malware, such as unique user ID (UID) for each application, system permissions, and its distribution platform Google Play. There are numerous ways to violate that fortification, and how the complexity of creating a new solution is enlarged while cybercriminals progress their skills to develop malware. A community including developer and researcher has been evolving substitutes aimed at refining the level of safety where numerous machine learning algorithms already been proposed or applied to classify or cluster malware including analysis techniques, frameworks, sandboxes, and systems security. One of the most promising techniques is the implementation of artificial intelligence solutions for malware analysis. In this paper, we evaluate numerous supervised machine learning algorithms by implementing a static analysis framework to make predictions for detecting malware on Android.

Key words: Supervised machine learning, classification, regression, data mining, obfuscation, security, Google Play, malware.

1. Introduction

In the last decade, a lot of machine learning and data mining based methods applied to detect intrusion, malware and the classification of them where many clustering and classification techniques implied to classify the malware into families and to also identify new malware families. Android is an open source mobile based operating system with more than one billion active users. This operating system is built on the Linux Kernel and integrates an architecture that is divided into five components. As security mechanisms Android makes use of two models of permissions, the first refers to a sandbox environment at the kernel level to prevent access to the file-system and other resources. The second model is a permissions API exposes to the user during an application to be installed. This tool allows requests to access some device resources [1] and [2].

Every application is assembled into an Android Application Package (APK) file includes application code (.dex files), resources, and the AndroidManifest.xml file that is a vital element provides the information of the features and the security configuration of every application including the information of the permissions API, activities, services, content providers and the broadcast receivers [3]. After Decompiling the APK file, we consider the AndroidMenifest.xml file for checking the permission used and then look into java file where the API functions are called. Also consider the asset and resource file to check is there any dex executable (ELF) image file or any code hiding image script available or not. In AndroidMenifest.xml file, permissions are used:

```
<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
<uses-permission android:name="android.permission.WAKE_LOCK" />
<uses-permission android:name="android.permission.READ_PHONE_STATE" />
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
<uses-permission android:name="android.permission.GET_ACCOUNTS"/>
<uses-permission android:name="com.google.android.gms.permission.ACTIVITY_RECOGNITION"/>
<uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED"/>
<uses-permission android:name="android.permission.CHANGE_WIFI_STATE" />
<uses-permission android:name="android.permission.GET_TASKS"/>
<uses-permission android:name="android.permission.ACCESS_BACKGROUND_SERVICE"/>
<uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED"/>
<uses-permission android:name="android.permission.SYSTEM_ALERT_WINDOW"/>
<uses-permission android:name="com.android.vending.BILLING" />
```

As per as security researcher, red marked permissions are commonly used in malicious android apps or games. We have also found some sensitive functions also called:

```
(TelephonyManager) context.getActivity().getSystemService("phone")).getDeviceId()
(TelephonyManager) context.getActivity().getSystemService("phone")).getLine1Number()
(ActivityManager) context.getActivity().getSystemService("activity")
List<RunningAppProcessInfo> procInfos = ((ActivityManager)
context.getActivity().getSystemService("activity")).getRunningAppProcesses();
for (int i = 0; i < procInfos.size(); i++) {
if (((RunningAppProcessInfo)
procInfos.get(i)).processName.equals(args[0].getAsString())) {
b = Boolean.valueOf(true);
}
}
String[] strArr = new String[]{"android.permission.ACCESS_COARSE_LOCATION",
"android.permission.ACCESS_FINE_LOCATION",
"android.permission.ACCESS_WIFI_STATE", "android.permission.CHANGE_WIFI_STATE",
"android.permission.VIBRATE", "android.permission.READ_CALENDAR",
"android.permission.WRITE_CALENDAR",
"com.google.android.gms.permission.ACTIVITY_RECOGNITION"};
IntentFilter intentFilter = new IntentFilter("android.intent.action.PACKAGE_ADDED");
intentFilter.addAction("android.intent.action.PACKAGE_REMOVED");
intentFilter.addAction("android.intent.action.PACKAGE_REPLACED");
intentFilter.addDataScheme("package");
```

```
registerReceiver(this.f2955D, intentFilter);
intentFilter = new IntentFilter();
intentFilter.addAction("android.net.wifi.SCAN_RESULTS");
intentFilter.addAction("android.net.wifi.STATE_CHANGE");
intentFilter.addAction("android.net.wifi.WIFI_STATE_CHANGED");
intentFilter.addAction("android.net.wifi.supplicant.CONNECTION_CHANGE");
registerReceiver(this.an, intentFilter);
```

During the last couple of years, the growth of malicious software on Android has enlarged due to several motives including the philosophy of Android, placing in the mobile market and the volume of sensitive information. There are numerous tools and techniques exists for examining threats for android operating system. Some of them are described below [1], [4]:

- Static analysis techniques assess malicious behavior in source code, data or binary files without running the application in a straight line. Its complexity has increased due to the practice assimilated by cybercriminals in the expansion of applications and it already shown that it is conceivable to avoid it using obfuscation techniques.
- Dynamic analysis techniques study the conduct of malware in execution by analyzing the running processes, the user interface, network connections, sockets opening through applying gesture simulation. Besides, previously exist methods allow to escape the process grasped by the dynamic analysis to stop its malicious behavior the malware can detect environments sandbox.
- Machine learning techniques projected to deliver systems capable to learn how to recognize malware without presence of programmed explicitly.

In this paper the results of several supervised machine learning methods including Support vector machines (SVM), Neural networks, Naïve Bayes, Decision Trees, Discriminant Analysis and K-Nearest neighbors (KNN) are assessed; in the classification of an application based on its permissions. Analyzing the outcomes experiment by these algorithms we found that the K-Nearest neighbors (KNN) algorithm has the best results in the classification prediction. Finally, the outcomes of this experiment aid as a source to propose new challenges in the design and development of new tools for the prediction to detect malicious software on Android.

The following sections of the article are composed of: Section 2 presents classification of android malicious application, Section 3 to Section 5 contains information about static analysis framework and the dataset used to the evaluation of algorithms, section 6 includes the results of the experiment, finally in section 7 include the conclusions and proposals for future work.

2. Classification of Android Malicious Application

We now briefly review classification of Android malicious application which motivates our work [5]-[7]:

String Obfuscation (Permission & function): To fingerprinting current methods of malware has started to use string-based features such as permissions and apps/package names including the declaration of application permissions, follow a strict syntax and must appear in the clear; other strings, such as names and identifiers, can be easily randomized or encrypted.

Native Code (Existence of executable file): To unburden malicious functionality from the main Dalvik Executable (DEX) to dynamic linked libraries (DLL) or other executable (ELF) files, Native code is recurrently used which are then invoked at runtime.

Dynamic Code Loading (Code injection from remote host/system): Native code and additional Dalvik Executable code are mostly loaded using a library from another app or from a remote system after being recovered at runtime. We found many examples of dynamic code loading where the code was loaded from outside of the app. Now numerous consistent software frameworks employ this technique, which is more attractive to malware writers, though the simple presence of dynamic code loading may not have malicious functionality.

Code Hiding (Bash/shell scripts extension change or hiding malicious code in image file uses a decoding algorithm to extract a malicious executable payload): To make the application look benign to cursory review the malware writers often proactively hide malicious components from the source code. The 'GingerMaster' malware is one of most who hides Bash scripts for its packaged root exploit under inoffensive file names in its resources like install.png and gbfm.png. Another malicious apps take an advance stepby using a technique of steganography by hiding malicious code inside a valid image file. The app practices a decoding algorithm to extract a malicious executable payload but loads the image through a outwardly benign action. Finally, Android malware can hide its malicious behavior in APK file hosted as a source of the main app. When the app gets executed the system dynamically loads the hidden component by installing the hidden APK.

Reflection (Execute in run time): Reflection, a common feature of Java frameworks, but it is also an infamous impediment to static analysis, if though it's difficult to determine statically that which code is getting executed at runtime. In a recent large-scale training, Lindorfer *et al*. showed that the general use of reflection among numerous android apps.

3. Related Works

It is necessary to do a study of the most appropriate machine learning algorithm because not all algorithm work efficiently for the same problem. In the analysis of detecting malware on Android is necessary to have a representative volume of applications both malicious and benign require a set of data for learning and testing. Through static or dynamic analysis, the challenge is to abstract the most appropriate evidence allowing an algorithm to classify the application as malicious or benign. There are some studies of the usage of machine learning algorithms with static and dynamic analysis as follows.

3.1. Machine Learning with Static Analysis

Urcuqui & Cadavid [8] propose a new module that implements a static analysis framework with six of machine learning algorithms including Naïve Bayes, Bagging, K-Neighbors, Support Vector Machines (SVM), Stochastic Gradient Descent (SGD) and Decision Tree to detect malware on Android and obtain the best result with the K-Neighbors algorithm in the classification task. To do experiment through the development of a permission scanner, the 558 AndroidManifest.xml was assessed in contrast to a list of 330 permissions with creating a binary vector that correspond to the permissions accessed by each application. Finally, the K-Neighbors algorithm presented the best results for the dataset was both in the classification (94% for benign software classification and 95% for malware) and in its performance (94%).

Sahs, J. and Khan, L. [9], proposed a supervised machine learning technique to detect malware on Android with the classification algorithm named Support Vector Machine (SVM) and static analysis as a technique to get the information of the Applications. During their experimentation, they used Androguard tool to extract information from the APKs and the Scikit-learn framework [10]. They created a binary vector containing the information of each permission used by every application and a control flow chart (CFG) from the Android Manifest file. The algorithm used a test set contained 2081 of benign and 91 of malicious applications including five kernels' information: one on binary vectors, one on chains, one on diagrams, a kernel for the sets, a kernel for common permissions and another for each application. Finally, they reached a low false negative rate but a false positive rate.

According to Ghorbanzadeh, M., Chen, Y., Ma, Z., Clancy, TC and McGwier, R. [11], where they proposed to assess the security vulnerabilities seeming in a structure of permissions of an Android application. They used a dataset of 1700 benign and malicious applications in their study, they also used an Apktool tool that allows decompiling an APK file to obtain .xml files like AndroidManifest.xml. They used 70% of data for training, 10% of data for validation and 20% of data for testing purpose. The Neural Network algorithm was applied with a binary vector representing the permissions requested by each application consists of two-layer structure with 10 neurons and an output layer of 34 neurons. Finally, they got an accuracy of 65%.

Yerima, S. Y., Sezer, S., McWilliams, G. and Muttik, I., proposed [12] one of a supervised machine learning method called Bayesian classifier, applied to learn the system containing (i) detector of calls to the API, (ii) detector of commands, (iii) detector of permissions, (iv) detector of encrypted code and (v) presence of secondary APKs or files.jar. In this experimentation every detector generated binary variables by analyzing APKs from the data with constructing a class to characterize the application as suspicious or benign. To do their work they used a set of 1600 application (800 for benign and 800 for malware) for training purpose, for the test purpose they used 400 applications (200 for benign and 200 for malware). Finally, the momentous outcome obtained TPR 90.6%, FNR 0.094%, accuracy of 93.5% and AUC of 97.22% for a total of 20 features and 1600 applications.

3.2. Machine Learning with Dynamic Analysis

DroidDolphin [13] is a dynamic analysis framework used in big data and machine learning to detect of malware on Android. Running the application on virtual environments its start analysis by extracting information from API calls and 13 activities. During this experimentation the supervised machine learning algorithm named SVM used the LIBSVM library [14]. The DroidDolphin used a set of 32000 benign and 32000 malicious applications for training purpose, 1000 of healthy and 1000 of malicious application for testing purpose, and got outcome a precision of 86.1% and an F-score of 0.875.

Feizollah, A., Anuar, N. B., Salleh, R., Amalina, F., Ma'arof, R. U. R., & Shamshirb and, S. [15] proposed a model by applying of five supervised machine learning classification algorithms (NB, KNN, DT, MLP and SVM). They used 100 samples of malware applications from the Android Genome Project and 12 samples of healthy applications. They extracted numerous information of the traffic of the network using dynamic analysis and detected the malicious activities from the Android devices. Finally, the machine learning obtained the results where KNN was the best algorithm with a TPR index of 99.94% against an RPF result of 0.06%.

3.3. Dataset

In 2012 a malicious dataset was published by a group of researches known as the Android Genome Project (MalGenome) [16]. In their experimentation they use 1260 malicious application with 49 dissimilar categories. Before publishing this dataset to the public, they characterized as a part outcome where 1083 of the malware (86.0%) were repackaged versions of genuine applications, 36.7% had the capability to raise privileges and 45.3% ware projected to premium messaging services subscribing. They got accuracy best for 79.6% and worst for 20.2% to detect malware with the evaluation of four security software.

4. Supervised Machine Learning

Based on evidence in the incidence of indecision, the supervised machine learning technique builds a model to make predictions using a known set of input data and responses by training a model to make predictions for the response to new data. A test dataset may also be used to validate that built model [17].

Supervised machine learning splits into two broad categories: (i) Classification (When the algorithm uses the data to predict a category), (ii) Regression (When a value is used to predict a continuous measurement for an observation).

Common classification algorithms [18]:

- Support vector machines (SVM)
- Neural networks
- Naïve Bayes classifier
- Decision trees
- Discriminant analysis
- K-Nearest neighbors (k-NN)

Common regression algorithms [18]:

- Linear regression
- Nonlinear regression
- Generalized linear models
- Decision trees
- Neural networks

Steps in Supervised Learning [19]:

1. Determine the type of training: Everyone needs to take decision for the selection of suitable data that is going to be used as a training set.

2. Gather a training set: A set of input objects and corresponding outcomes should be gathered from the real-world uses such as human experts or from measurements.

3. Determine the input feature representation of the learned function and structure: The number of features within the data should contain adequate information to correctly predict the output because the accuracy of the learned function is fully depended on the representation of input object.

4. Complete the design: To run the learning algorithm using collected training set some supervised learning algorithms necessitate the user to regulate certain parameters to optimize the performance of the training set.

5. Evaluate the accuracy of the learned function: Measure the performance of the outcome of the function using a test set separated from the training.

5. Methodology

In this section, several corresponding experiments are conducted to estimate the performance of the supervised machine learning algorithms based on the Android's configuration permission for the assessment of the outcomes. In our methodology section we build a model using a study of static analysis of the application configuration permissions containing train and test datasets, assessment standards, the outcomes of the experimentation of these algorithms. The dataset can be described as Table 1:

Table 1. Dataset Description

	Android Malware		Training Set		Test Set	
Value	Count	Percent	Count	Percent	Count	Percent
0	199	50.00%	116	48.54%	83	52.20%
1	199	50.00%	123	51.46%	76	47.80%

5.1. Implementation of Static Analysis Framework Using Supervised Learning

The proposed framework is divided into five steps:

1. Gather a training set: A total of 558 APKs were collected where 279 low-privilege applications of the open source dataset and a random selection of 279 MalGenome malware.

2. Extraction: To attain the AndroidManifest.xml file from the collected APK we used the ApkTool 2.3.0 Tool (Released-21 Sep 2017).

3. Features vector: We used MATLAB (2017a) for the development purpose with creating the binary vectors by accessing of every application's permission, and a label of application classification was added to determine whether the application as benign or malicious. A new dataset also created using the previous outcomes containing the permissions accessed by each application in binary. Each application is allotted a vector defined by V = (R1, R2, ..., R330, C) containing the information of the permissions using.

$$R_i = \begin{cases} 1, & \text{if detected an access permission} \\ 0, & \text{in another case} \end{cases}$$

$$C_i = \begin{cases} 1, & \text{if the application is malicious} \\ 0, & \text{in another case} \end{cases}$$

4. Training and testing: For our experimentation, we divided the dataset into two sections where 60% of the data used for training and 40% of the data, selected randomly, for testing purpose. Finally, we trained and verified our experiment by each supervised machine learning algorithms (Support vector machines (SVM), Neural networks, Naïve Bayes, Decision Trees, Discriminant Analysis and K-Nearest neighbors).

5. Classification: After training and testing using these supervised learning algorithms we reached at position with classifying the applications as benign or malicious shown in Fig. 1. So that the user can take decision to install that application on their Android devices.

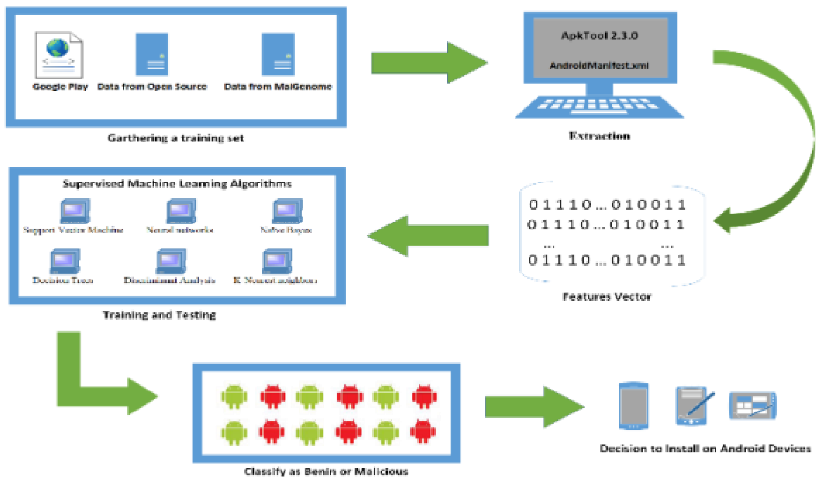

Fig. 1. A static analysis framework to detect malware on Android.

5.2. Measurement Metrices

Confusion Matrix: A confusion matrix is a matrix which contains information about actual and predicted classifications in Table 2 done by a classification to measure numerous algorithm performance using the matrix data [20], [21].

Table 2. Confusion Matrix

		Actual class	
		Positive	**Negative**
Predicted Class	**Positive**	True Positive (TP)	False Positive (FP)
	Negative	False Negative (FN)	True Negative (TN)

- Accuracy (AC) is the proportion of the total number of corrected predictions. Overall, how often is the classifier correct?

$$\text{Accuracy} = \frac{\text{No. of correctly classified data (TP + TN)}}{\text{Total}}$$

- Misclassification Rate (1-Accuracy) describe the overall, how often is it wrong?

$$\text{misclassification} = \frac{\text{No. of misclassified data (FP + FN)}}{\text{Total}}$$

- Precision (P) is the proportion of the correctly predicted positive cases determined by:

$$\text{precision} = \frac{\text{TP}}{\text{TP + FP}}$$

- The recall or true positive rate (TP) is the proportion of the correctly identified positive cases defined by:

$$\text{recall} = \frac{\text{TP}}{\text{TP + FN}}$$

- f1- Score is a weighted average of the True Positive (TP) rate (recall) and Precision (P).
- ROC Curve is a graph to summarize the performance of the classifier over all probable thresholds generated by plotting the True Positive (TP) Rate in Y-axis against the False Positive (FP) Rate in X-axis.

6. Experimentation and Result

6.1. Technologies Used

The experiments are done using MATLAB (2017a) on the machine where the System Model was MacBookPr011.5, Processor was Intel(R) Core(TM) i7-4870HQ CPU @ 2.506Hz (8 CPUs), ~2.SGHz 64-bit PC.

6.2. Result and Discussion

The supervised machine learning algorithms used in our experiment including Support vector machines (SVM), Neural networks (NN), Naïve Bayes (NB), Decision Trees (DT), Discriminant Analysis (DA) and K-Nearest neighbors (kNN). After applying these algorithms, the results got from the test are summarized in Table 3.

Table 3. Result of the Performance of Supervised Machine Learning Algorithms

Algorithms	Precision		Recall		Misclassification rate		Accuracy
	0	1	0	1	0	1	
NB	0.85	0.87	0.82	0.83	0.15	0.17	0.84
DA	0.81	0.84	0.89	0.89	0.14	0.15	0.86
kNN	0.96	0.96	0.97	0.96	0.04	0.04	0.96
DT	0.93	0.93	0.93	0.92	0.06	0.05	0.94
NN	0.87	0.90	0.85	0.89	0.12	0.10	0.89
SVM	0.91	0.93	0.92	0.94	0.08	0.06	0.93

From the results we get, Naive Bayes and Discriminant analysis give the worst results both in the classification for benign and malicious behavior and the performances of them are 84% and 86% respectively. The neural network algorithm gives the average result with 89% performance. Decision Tree and SVM algorithms provide almost similar results with 94% and 93% performances respectively. Finally, the algorithm that presented the best results for our dataset was K-Nearest Neighbors both in the classification (96% for benign software classification and 96% for malware) and overall performance is 96%. The above results can be seen through the ROC curve (Fig. 2) and the misclassification curve (Fig. 3).

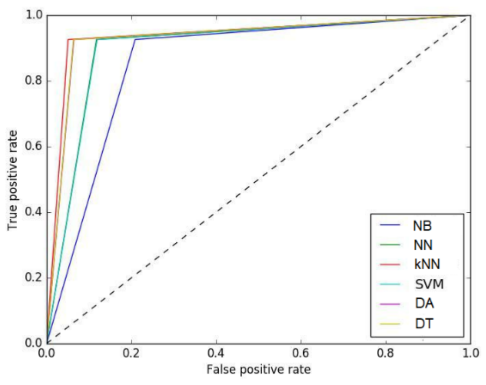

Fig. 2. ROC curve.

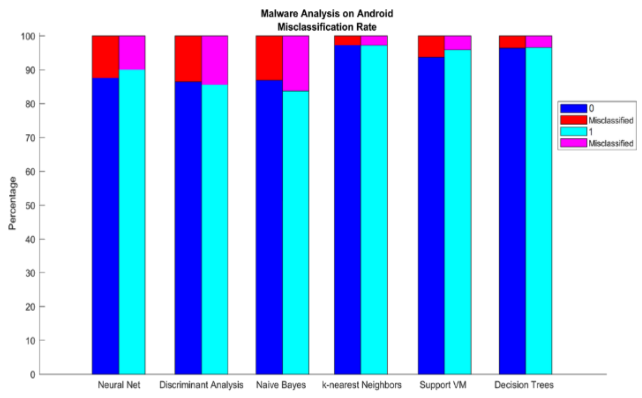

Fig. 3. Misclassification rate.

7. Conclusion and Future Work

In this article we propose the assessment numerous supervised machine learning algorithm to detect malware on Android from their access permissions. From the experiment we can show that, the assessment of our dataset presents as best classification algorithm K-Neighbors. As future work suggestions, we will try to do the following:

- Assess trained classifiers in contradiction of another set of vigorous applications with over privileges.
- Study and Train other artificial intelligence techniques.
- Implement a system that will allow the algorithm to learn from wicked classifications.
- Discover other features of Android for the training and testing.

References

[1] Urcuqui, C., & Navarro, A. (2016, August). Framework for malware analysis in Android. *Sistemas & Telemática, 14(37)*, 45-56.

[2] Drake, J. J., Lanier, Z., Mulliner, C., Fora, P. O., Ridley, S. A., & Wicherski, G. (2014, March). *Android Hacker's Handbook*. Indianapolis: John Wiley & Sons.

[3] Peiravian, N., & Zhu, X. (2013, November). Machine learning for android malware detection using

permission and API calls. *Proceedings of the 2013 IEEE 25th International Conference on Tools with Artificial Intelligence (ICTAI)* (pp. 300-305).

[4] Suarez-Tangil, G., Dash, S. K., Ahmadi, M., Kinder, J., Giacinto, G., & Cavallaro, L. (2017, March 22-24). DroidSieve: Fast and accurate classification of obfuscated android malware. *Proceedings of the ACM Conference on Data & Application Security and Privacy*. Scottsdale, AZ, USA.

[5] Batyuk, L., Herpich, M., Camtepe, S. A., Raddatz, K., Schmidt, A., & Albayrak, S. (2011). Using static analysis for automatic assessment and mitigation of unwanted and malicious activities within Android applications. *Proceedings of the 6th IEEE International Conference on Malicious and Unwanted Software (MALWARE).* Piscataway.

[6] Sharif, M. I., Lanzi, A., Giffin, J. T., & Lee, W. (2008, February). Impeding malware analysis using conditional code obfuscation. In *NDSS, the Internet Society*. Springer Berlin Heidelberg.

[7] Petsas, T., Voyatzis, G., Athanasopoulos, E., Polychronakis, M., & Ioannidis, S. (2014, April). Rage against the virtual machine: Hindering dynamic analysis of android malware. *Proceedings of the Seventh European Workshop on System Security* (p. 5). ACM.

[8] Urcuqui, C., & Navarro, A. (2016, April). Machine learning classifiers for android malware analysis. *Proceedings of the 2016 IEEE Colombian Conference on Communications and Computing (COLCOM)* (pp. 1-6). IEEE.

[9] Sahs, J., & Khan, L. (2012, August). A machine learning approach to android malware detection. *Proceedings of the 2012 European Intelligence and Security Informatics Conference (EISIC)* (pp. 141-147). IEEE.

[10] Scikit-Learn 0.16.1 Documentation. Scikit-learn: Machine learning in Python. Retrieved from the website: http://scikit-learn.org/stable/

[11] Ghorbanzadeh, M., Chen, Y., Ma, Z., Clancy, T. C., & McGwier, R. (2013, January). A neural network approach to category validation of android applications. *Proceedings of the 2013 International Conference on Computing, Networking and Communications (ICNC)* (pp. 740-744). IEEE.

[12] Yerima, S. Y., Sezer, S., McWilliams, G., & Muttik, I. (2013, March). A new android malware detection approach using bayesian classification. *Proceedings of the 2013 IEEE 27th International Conference on Advanced Information Networking and Applications (AINA)* (pp. 121-128).

[13] Wu, W. C., & Hung, S. H. (2014, October). DroidDolphin: A dynamic Android malware detection framework using big data and machine learning. *Proceedings of the 2014 Conference on Research in Adaptive and Convergent Systems* (pp. 247-252). ACM.

[14] Chang, C. C., & Lin, C. J. (2011). LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology (TIST)*, *2(3)*, 27.

[15] Feizollah, A., Anuar, N. B., Salleh, R., Amalina, F., Ma'arof, R. U. R., & Shamshirband, S. (2014). A study of machine learning classifiers for anomaly-based mobile botnet detection. *Malaysian Journal of Computer Science, 26(4)*.

[16] Krutz, D. E., Mirakhorli, M., Malachowsky, S. A., Ruiz, A., Peterson, J., Filipski, A., & Smith, J. (2015 May). A dataset of open-source Android applications. *Proceedings of the 2015 IEEE/ACM 12th Working Conference on Mining Software Repositories (MSR)* (pp. 522-525). IEEE,

[17] Supervised Learning Workflow and Algorithms. Retrieved from the website: https://www.mathworks.com/help/stats/supervised-learning-machine-learning-workflow-and-algorithms.html

[18] Supervised Learning. Retrieved from the website: https://www.mathworks.com/discovery/supervised-learning.html

[19] Supervised Learning. Retrieved from the website: https://en.wikipedia.org/wiki/Supervised_learning

[20] Confusion Matrix. Retrieved from the website: http://www2.cs.uregina.ca/~dbd/cs831/notes/confusion_matrix/confusion_matrix.html

[21] Dataschool. Simple guide to confusion matrix terminology. Retrieved from the website: http://www.dataschool.io/simple-guide-to-confusion-matrix-terminology/

Md. Shohel Rana was born in Sirajganj, Rajshahi, Bangladesh, in 1987. He received his B.Sc. (Engg.) and M.Sc. (Engg.) both in computer science and engineering from the Mawlana Bhashani Science and Technology University, Tangail, Bangladesh in 2010 and 2014 respectively. Currently he is pursuing his Ph.D. in computational science under School of Computing at The University of Southern Mississippi, Mississippi, United States. His research area includes digital image processing and computer vision, data science, machine learning, human computer interaction, e-learning, remote learning, distributed database system and web technology.

Andrew H. Sung is the director of the School of Computing and professor at The University of Southern Mississippi, United States. He received his Ph.D. in computer science from State University of New York at Stony Brook, United States, 1984. His research area includes computational intelligence and its applications, information security and assurance, bio- and medial-informatics, data mining and pattern recognition, high-performance computing, petroleum exploration and reservoir modeling.