A STUDY OF MACHINE LEARNING CLASSIFIERS FOR ANOMALY-BASED MOBILE BOTNET DETECTION

Ali Feizollah[1], Nor Badrul Anuar[2], Rosli Salleh[3], Fairuz Amalina[4], Ra'uf Ridzuan Ma'arof[5], Shahaboddin Shamshirband[6]

[1, 2, 3, 6] Security Research Group (SECReg), University of Malaya, 50603, Kuala Lumpur, Malaysia
[4] Mobile Cloud Computing (MCC), University of Malaya, 50603, Kuala Lumpur, Malaysia
[2, 3] Faculty of Computer Science and Information Technology, University of Malaya, 50603, Kuala Lumpur, Malaysia
[5]F-Secure Corporation, Malaysia
[6] Department of Computer Science, Islamic Azad University, Chalous Branch, P.O. Box 46615-397, Chalous, Iran

[1]ali.feizollah@siswa.um.edu.my, [2]badrul@um.edu.my, [3]rosli_salleh@um.edu.my
[4]fairuzamalina@siswa.um.edu.my, [5]rauf.ridzuan@f-secure.com

ABSTRACT

In recent years, mobile devices are ubiquitous. They are employed for purposes beyond merely making phone calls. Among the mobile operating systems, Android is the most popular due to its availability as an open source operating system. Due to the proliferation of Android malwares, it is crucial to study the best classifiers that can detect these malwares effectively and accurately through selecting the most suitable network traffic features as well as comprehensive comparison with related works. This study evaluates five machine learning classifiers, namely Naïve Bayes, k-nearest neighbour, decision tree, multi-layer perceptron, and support vector machine. The evaluation was validated using malware data samples from the Android Malware Genome Project. The data sample is a collection of malwares gathered between August 2010 and October 2011 by the University of North Carolina. Among various network traffic characteristics, three network features were selected: connection duration, TCP size and number of GET/POST parameters. From the experiment, it is found that k-nearest neighbour provides the optimum results in terms of performance among the classifiers. The experimental results also indicate a true positive rate as high as 99.94% and false positive of 0.06% for the k-nearest neighbour classifier.

Keywords: Mobile botnet, machine learning classifiers, anomaly-based detection, intrusion detection systems

1.0 INTRODUCTION

Mobile devices have become an inseparable component of most people's lives, replacing personal computers in terms of the Internet usage by allowing users to check emails, access online banking services, tweet, or use Facebook on such devices. Furthermore, the rapidly growing rich mobile applications [1] with overwhelming user experience, such as maps and GPS functions, make mobile devices more appealing to users. As part of utilizing mobile devices, certain sensitive data such as contact lists, passwords and credit card numbers are stored on these mobile devices. Based upon this scenario, hackers have turned their attention to mobile devices where it is possible to obtain an abundance of their preferred data, whereby security issues are taken less seriously on such devices.

Furthermore, an infected mobile device, known as a bot, can become a component of a large network of infected mobile devices, which is referred to as a botnet. A botnet is controlled by a hacker, who is called a Botmaster. As an example, RootSmart collects a wide range of information and sends it to Botmasters [2]. Basically when the device is infected, the bot (malware) contacts and informs the Botmaster that the device has successfully been infected. The connection between a bot and its Botmaster can be done in three ways. First, a chat server may be specified to make the connection in such a way that the bot and a Botmaster log in to the chat server and communicate like two persons. Second, a web server known as command and control (C&C) server works as a link between the bot and the Botmaster. The bot employs HTTP traffic to get to the web server. It merges with

the HTTP traffic to make detection more difficult. This communication mode is the most prevalent and is the focus of this paper. The third is the P2P mode where the botmaster contacts one or a few selected bots which in turn will contact other bots. A botnet can be utilized for spamming, sending premium short message service (SMS) on a large scale and distributed denial of service (DDoS) attacks, and stealing users' data. For instance, DroidDream [2] gains root permission and installs another application that prevents the malware's removal. It has a silent pattern whereby it activates quietly, at night, when the user is asleep [2].

Among the various mobile operating systems, Android has experienced more attacks since it is an open source operating system [3]. In addition, users are supposed to download and install an application from the Android market, but some applications were installed from unofficial Android markets, like SlideME market [4]. Surprisingly, the official Android Market does not firmly control its applications. As an example, according to security researchers, in April 2013, some malware started operating inside Google Play for at least 10 months, infecting 35 different applications [5]. The lookout mobile security corporation reported that the infected applications had been downloaded between 2 and 9 million times [5]. Therefore, developing an effective intrusion detection system (IDS) remains a challenge but is imperative to billions of mobile users.

Although Android malware detection has been meticulously studied, little attention has been directed to the network traffic generated by malwares. In this paper, we evaluate machine learning classifiers on network traffic to detect malicious activities on mobile devices. An advantage of machine learning classifiers is that the computational burden is relatively small, thus the results of this study may potentially help to develop an intrusion detection system (IDS) for analyzing real-time network traffic. In addition, the use of machine learning classifiers is proven to enhance detection accuracy [6].

In this paper, 5 types of machine learning classifiers were used on the data sample, i.e. Naïve Bayes (NB), k-nearest neighbour (KNN), decision tree (DT), multi-layer perceptron (MLP), and support vector machine (SVM). The results were collected based on performance, with regards to detection rate and false positive rate and speed in terms of the time taken to build a detection model - since this study is on mobile devices with limited resources.

The rest of the paper is organised as follows. Section 2 reviews a number of related works. Section 3 outlines the methodology including the various processes, methods, descriptions of different components and machine learning classifiers employed. Section 4 tabulates the experimental results and presents a discussion which addresses the effectiveness, performance and receiver operating characteristic (ROC) curve analysis. Finally, Section 5 concludes the paper.

2.0 RELATED WORKS

Machine learning is a branch of artificial intelligence dealing with teaching machines how to make decisions. It has been used for years to develop intelligent systems. For instance, machine learning was used by Shamshirband et al. [7] for rescue purposes. Similarly, Feizollah et al. [8] used combination of machine learning to detect DDOS attack. Machine learning is provided with a labelled dataset, and a model is produced as output, which can be applied to new data. The classifiers learn from several labelled inputs to build a model - analogous to the way a child learns to identify a dog from examples of dogs, after which he/she is able to distinguish a dog from all kinds of animals. Therefore, due to their learning capability, machine learning classifiers were selected for this study.

Two methods applied in malware detection are misuse-based and anomaly-based. The misuse-based methods are also known as signature-based methods, and they are mainly used by antivirus software that relies on detecting malware based on unique signatures. Although it is very precise, it is of no value against unknown threats and it requires constant signature updates [9]. Yajin and Xuxian [10] demonstrated that traditional anti-virus software are able to detect malware up to 79.6%. Droid Analytics [11] is an Android malware detection system, which automatically collects, extracts and analyses the signature of the Android application file. It extracts methods and classes from the application's Java code and employs them as signatures. Subsequently, the generated signatures are used to detect malicious applications. Nonetheless, the aforementioned method is useful only for known malwares, whereas with the advent of new threats, the same process must be performed and the generated signature has to be added to the database. Anomaly-based methods, on the other hand, depend on classifiers to train a system to differentiate between normal and malware behaviour, which can be used to

detect anomalies so as to discover unknown malwares. Sahs and Khan [12] extracted a number of features and applied support vector machine (SVM) to them to make the system learn the pattern of malicious applications against normal ones. The results were highly accurate, with 93% of accuracy. DroidMat is another example of machine learning in malware detection [13]. Wu et al. [13] used K-means to inspect an application, obtaining 87.39% detection accuracy as opposed to the misuse-based methods mentioned earlier with only 79.6%. Hence, the anomaly-based detection techniques were chosen for the purpose of this study because they are capable of detecting anomalies based on what they learn.

There are two types of malware analysis: static analysis and dynamic analysis. Static analysis is the examination of an Android installation file known as APK to detect suspicious applications. For instance, a study was conducted in which the requested and required permissions were inspected to detect Android malware [14]. However, the problem with static analysis is that some malwares such as DroidKungFuUpdate stealthily download malicious codes [10]. Thus, the malicious code is undetectable via static analysis [10]. On the other hand, dynamic analysis tries to identify malwares based on their behaviour. For one, Crowdroid [15] collects the device's kernel system calls and sends them to a remote server for processing. By using system call as one of the features, malware can be detected based on similar behaviours and patterns. For that reason, the present study concentrates on using dynamic analysis with emphasis on malware network behaviour.

Dynamic analysis is the analysis of an application's behaviour. Two of the most important behaviours used in the dynamic analysis are system calls and network traffic. When an application is running, in order to perform tasks, it should request for some operations from an operating system, such as read, write, or open. Therefore, if an application was calling too many functions, it would sound suspicious. Crowdroid [15] focused on collecting system calls and processing them to detect an anomaly. Since the Android operating system has the Linux kernel [15], collecting system calls is a complicated task as described in [15]. In most cases, the device must be rooted, which is disabling part of the operating system's security architecture, consequently leaving the device more vulnerable against threats. On the other hand, collecting network traffic is done by an application such as tPacketCapture Pro [16] on the device. Su et al. [17] used network traffic with machine learning classifiers to detect malware in Android. With decision tree and random forest, they were able to detect 91.60% and 96.70% of the malicious traffic, respectively.

3.0 METHODOLOGY

This paper aims to study the best classifier for detecting Android malware using machine learning classifiers in order to confront the malwares rapid growth. Fig 1 illustrates the details of the study workflow. There are three main processes: data collection (i.e. the normal and infected traffic), feature selection and extraction, and machine learning classifiers. In the data collection process, normal and infected traffic are collected separately. In the next process, the selected network characteristics are extracted for inspection by the classifier. Normal and infected data are then combined, randomized and labelled. In the final process, the prepared dataset is fed to 5 classifiers and the results are compared with two of the related works.

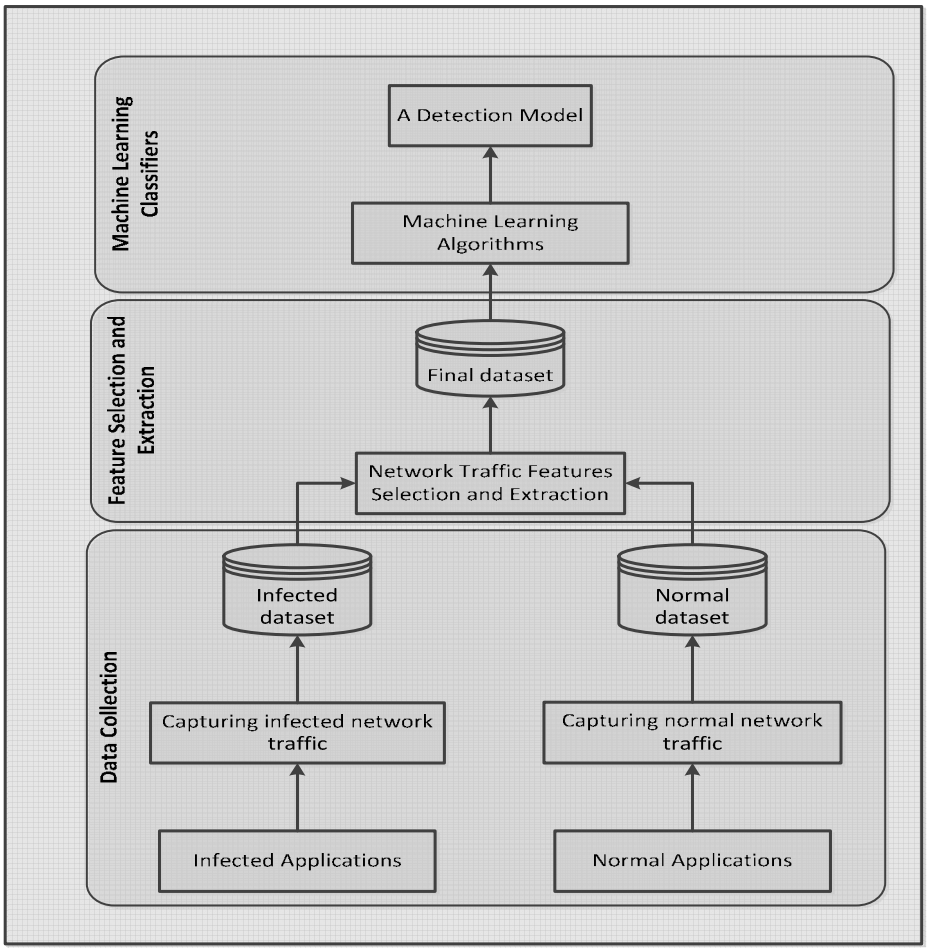

Fig 1. Process Flow

3.1 Data Collection

The MalGenome [10] data sample was utilized for the current work, which includes 1260 infected Android samples in 49 different families. A malware family is basically a collection of malware presenting similar behaviour. Among these samples, the most widespread ones, or 10% of all data samples, were selected for this study. It is worth mentioning that 93% of the MalGenome samples are botnets inclusive of the ones selected in this study.

Data samples are merely installation files that are not usable for research works. They have to be converted to a dataset that is series of data collected from analysing the data samples. Thus, Anubis [18], CopperDroid [19] and SandDroid [20], which are dynamic analysis platforms for Android, were utilized to collect network traffic generated by each sample. Table 1 represents the families chosen for this experiment.

Table 1. List of Android Malware Families Used

No.	Malware Family Name	No. of Samples	Discovered Month	Characteristics
1	AnserverBot	13	September-11	Silently downloads an update for malicious application on run time containing malicious code from a hacker
2	BaseBridge	11	June-11	Silently updates malicious application and downloads a malicious code from a hacker
3	DroidDream	11	March-11	Hijacks an application and controls the UI and performs commands received from a hacker
4	DroidKungFu3	10	August-11	Malicious code is encrypted and it steals user's phone number and send it to hacker
5	DroidKungFu4	10	October-11	C&C server address is in the native program but in cipher text. It receives commands from a hacker
6	Geinimi	9	December-10	Makes phone calls in background and sends premium sms. Commands are received from a hacker
7	GoldDream	10	July-11	Makes phone calls in background and sends premium sms. Commands are received from a hacker
8	KMin	10	October-11	Sends premium sms without user's knowledge. Commands are received from a hacker
9	Pjapps	9	February-11	Sends premium sms without user's knowledge and steals user's phone number. Sends stolen data to a hacker
10	Plankton	7	June-11	Downloads malicious code as an update from a hacker
	Total	100		

Apart from analysing infected samples, 12 normal application samples were examined and their network traffic was collected on a device by installing the most popular applications from the Android market. tPacketCapture Pro [16] was purchased from Google Play for the network traffic collection process. The paid version of the application is capable of monitoring specific applications as specified by the user, and capturing network traffic as opposed to the free version, which captures all of a device's traffic. Such a feature in the paid version assures data validity, since it is essential to have the network traffic of only one specific application. Each of the normal application was run between 10 to 15 minutes so as to capture its patterns. The device and the applications were clean. After that, selected network traffic features were extracted using tshark [21] (tshark is a command line application similar to Wireshark). Table 2 shows the normal applications chosen from the Android market.

Table 2. Normal Applications Selected from Google Play

Facebook	MailDroid
Twitter	Feedly
Chrome	Maps
Google Reader	Gmail
Flipboard	Skype
Youtube	Facebook Messenger

Consequently, the final dataset was a combination of normal and infected traffic. The data was randomly arranged in the dataset using a pre-processing feature in the Waikato Environment for Knowledge Analysis (WEKA), which is a machine learning software, and in particular a package called *weka.filters.unsupervised.instance.randomize*. This way, the classifier training process is more accurate. Table 3 presents an excerpt from the labelled dataset.

Table 3. The Dataset Sample

tcp_size	duration	no_parameter	results
0	0.432149	0	normal
166	0.513963	0	normal
306	0.137004	9	infected
448	0.317172	0	normal
347	0.080052	6	infected
0	0.00002	0	normal
1164	0.050994	1	infected
0	0.000038	0	normal
0	0.634737	0	normal
306	0.007583	9	infected
0	2.859491	0	normal
0	7.967501	0	normal
306	0.051335	9	infected
166	0.709343	0	normal
306	0.035025	9	infected
0	6.209529	0	normal
807	0.386469	10	infected
0	0.000541	0	normal
197	4.230825	0	normal
0	6.00186	0	normal
448	2.549219	9	normal
0	0.000011	8	normal
347	0.051966	6	infected
0	0.001416	0	normal
318	0.19875	1	normal

3.2 Feature Extraction

Among numerous features of network traffic, only 3 features were selected. As the number of selected features rises, the processing power and processing time soar. The features are as follows:

- **Connection duration**: it describes how long a connection lasts. Technically, an http-based bot is not constantly connected to the server. It connects to the server at relatively specific intervals to check whether there is a new command from the hacker (botmaster) or not. Thus, most of the communications consist of simple handshakes, and it is a plausible feature for malware detection.
- **TCP size**: one of the most essential functions of a mobile bot is to steal user information and send it to a hacker. For that purpose, TCP payload includes mobile device data with distinguishable size from other packets. Thus, TCP size was chosen for this work.
- **Number of parameters in GET/POST request**: GET and POST are two methods in http protocol used to submit data from the client to the server. They should not literally be considered as English words. For instance, Fig 2 shows a POST method from Geinimi malware on a mobile device leaking user data to a C&C server.

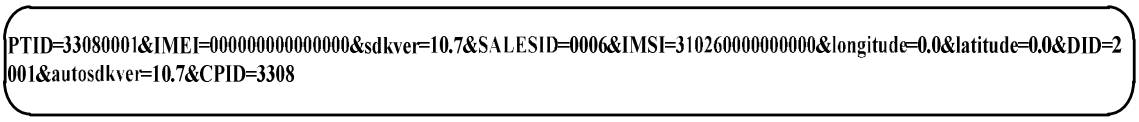

Fig 2. Leaked data from malware to a server

Fig 2 shows that 10 different kinds of information or parameters are sent from the user's device to a server by Geinimi malware, namely PTID, IMEI, sdkver, SALESID, IMSI, longitude, latitude, DID, autosdkver and CPID. However, in normal network traffic on a mobile device, such pattern is seen occasionally. Hence, the number of parameters is a possible candidate for our analysis and to our knowledge it has not been used in any previous papers.

3.3 Machine Learning Classifiers

The classifiers work with a labelled dataset and find a pattern to build a proper detection model. In this study, 5 different classifiers have been used, ranging from a simple to more complex and powerful ones, as follows:

- **Naïve Bayes:** It is a simple probabilistic classifier based on the Bayes theorem with a strong features independence assumption, meaning that, it presumes there is no dependency between various dataset features, something that is rarely true [22].
- **K-nearest Neighbor:** It is one of the most straightforward classifiers, also referred to as KNN [23]. Regardless of its simplicity, it has accomplished a number of pattern recognition tasks [24]. In this project, 3 neighbours were chosen to perform this classifier.
- **Decision Tree (J48):** It is a renowned, relatively simple classifier. It is an open source Java implementation of the C4.5 decision tree. The model looks like a tree and a decision is made based on whether a record of data belongs to a branch or not. It is a popular classifier since it is easy to interpret and explain [25].
- **Multi-Layer Perceptron (MLP):** The multi-layer perceptron (MLP) is a type of artificial neural network (ANN) consisting of a network of neuron layers inspired by the human brain. It has been widely employed among researchers in various fields such as banking, defence, and electronics [26]. MLP has medium-level complexity.
- **Support Vector Machine (SVM):** The support vector machine was developed in the reverse order of a neural network. It has a robust theoretical background, which renders it superior in terms of performance compared to neural networks. However, it is complex, hard to interpret, CPU-bound, and memory-intensive [27].

In order to evaluate each classifier, 2 validation methods known as *k*-fold cross-validation and 70% split were used. The *k*-fold cross-validation method is a means of enhancing the holdout method. The data set is divided into *k* subsets, and the holdout method is repeated *k* times. Each time, one of the *k* subsets serves as the test set and the other *k*-1 subsets are compiled to form a training set. Then, the average error across all *k* trials is computed. The advantage of this method is that it matters less how the data gets divided. Every data point gets to be in a test set exactly once, and in a training set *k*-1 times. The variance of the resulting estimate is reduced as *k* increases. The disadvantage of this method is that the training phase has to be rerun from scratch *k* times, meaning it takes *k* times as many computations to perform an evaluation [28]. Specifically, a 10-fold option was used, which is described as applying the classifier to data 10 times and every time with a 90-10 configuration, i.e. 90% of data for training and 10% for testing. The final model is the average of all 10 iterations.

The second method is 70% split, which is defined as using 70% of a dataset for training purposes. The benefit of this method is that it takes much less time compared to the 10-fold method since the process is done once, whereas the same process is done 10 times for the other method. Over-fitting is a drawback of the 70% method, and it occurs when a classifier memorizes a dataset instead of getting trained. Generally, in the majority of the experiments, the 10-fold method produces better results than the split method [28].

This experiment was performed on a desktop computer with Intel core i5 2400 CPU at 3.10 GHZ and 4GB of RAM. The operating system of this machine is Microsoft Windows 7. The WEKA machine learning software

was used for this study due to its simplicity and user-friendly environment [29]. The Java Runtime Environment (JRE) version 1.7.0_21 ran the WEKA software.

4.0 RESULTS AND DISCUSSION

It is important to reiterate that the purpose of this work is to study and evaluate various machine learning classifiers for mobile malware detection in order to tackle the problem of rapid growth of malwares. In intrusion detection and response studies, detection rate (i.e. true positive and false positive rates), precision and recall are examples of system performance measurements [30].

4.1 Effectiveness

The results are expressed in terms of performance measurements. Detection rate, also known as a true positive rate (TPR), is the probability of correctly detecting an instance as malware. Additionally, false positive rate (FPR) is another measurement defined as the false detection of normal traffic as infected. The higher the TPR, the better the result is. Conversely, the lower the FPR, the better it is. The empirical results are presented in Table 4.

Table 4. The Experimental Results

	10-fold validation		**70% split validation**	
	TPR	**FPR**	**TPR**	**FPR**
Naïve bayes	93.00%	7.00%	93.13%	6.86%
K-nearest neighbor (KNN)	99.94%	0.06%	99.53%	0.46%
Decision Tree (J48)	99.70%	0.30%	99.20%	0.80%
MLP	97.04%	2.96%	97.33%	2.66%
SVM	99.50%	0.50%	99.33%	0.66%

One of the purposes of this work is to study the detection methods. As a result, performing a comparison with other studies is common in the research community to demonstrate improvements over related works. As mentioned earlier, a study was conducted by Su et al.[17] in which the MalGenome data sample was used with classifiers. In this paper, 100 malwares from the MalGenome data sample were used, whereas Su et al. [17] used 49 from the same data sample. In terms of number of classifiers, 5 classifiers were used in this work, ranging from simple to complex classifiers, but Su et al.[17] used only 2 classifiers. More classifiers facilitate a more comprehensive analysis. As discussed previously, in this work 3 of the most effective network traffic features were applied, namely connection duration, TCP size and number of GET/POST parameters. On the other hand, Su et al.[17] employed 9 features, namely the average and standard deviation of the number of sent/received packets, the average and standard deviation of the number of bytes sent/received and the average TCP/IP session duration. It should be noted that as the number of features increases, the classifiers need to process additional data, which leads to higher processing power consumption as well as longer time. The decision tree (J48) and random forest classifiers utilized by Su et al. [17] produced true positive rates of 91.6% and 96.7% respectively. However, as Table 4 depicts, the best result in this work was achieved with the KNN classifier, with as much as 99.94%. It was discussed in Section 3.3 that the 10-fold validation usually produces enhanced results. In this study, the best result of the 10-fold validation is 99.94% while the best result of the 70% split validation is 99.53%.Therefore, a comparison between the 2 studies concurs that an improvement has been achieved in this research work through selecting the most suitable network traffic features as well as increasing the detection rate. The table below summarizes the outcome of the two studies.

Table 5. Result Comparison with Similar Work

	Su et al. [15]	Current study	
		10-fold validation	70% split validation
Naïve bayes	-	93%	93.13%
K-nearest neighbor (KNN)	-	99.94%	99.53%
Decision Tree (J48)	91.60%	99.70%	99.20%
Random Forest	96.70%	-	-
MLP	-	97.04%	97.33%
SVM	-	99.50%	99.33%

According to the above table, it is found that the results of this study are superior to similar work done by Su et al. [17]. A 99.94% detection rate was achieved in this study using the K-nearest neighbour classifier compared to 96.7% in the other work using the random forest classifier.

4.2 Performance

To reiterate, it should be mentioned that the output of the training phase is a detection model. Since this study was conducted on mobile devices, time is an important factor. As the processing time of a classifier goes up, the more resources, such as CPU, battery, and memory are solicited in a mobile device by the classifier. Time is defined as how long it takes to prepare a final model in the training phase. The emphasis is on the 10-fold method time as it is more logical compared to the 70% split method. Because the 10-fold validation is done 10 times on data and the average is the last result, the 70% method applies an algorithm on data only once and it is considered a very positive aspect. The k-nearest neighbour classifier produced the best time result with 0.01 seconds. Naïve Bayes and the decision tree are second and third with 0.04 and 0.09 seconds respectively. In the fourth and fifth place are SVM and MLP with 0.31 and 1.80 seconds accordingly. The timing results for both validation methods are tabulated in Table 6.

Table 6. Comparison of Processing Time of Classifiers (in seconds)

classifiers	10-fold cross-validation	70% split validation
Naïve bayes	0.04	0.01
K-nearest neighbor (KNN)	0.01	0.01
Decision Tree (J48)	0.09	0.03
MLP	1.8	1.84
SVM	0.31	0.15

As a conclusion, the k-nearest neighbour exhibits optimal performance in terms of detection rate. With respect to time, the k-nearest neighbour is the first among other classifiers.

4.3 ROC Curve

A Receiver Operating Characteristic (ROC) curve is normally used to measure intrusion detection performance. It indicates how the detection rate changes, as the internal threshold is varied to generate more or fewer false alarms. It plots intrusion detection accuracy against false positive probability. Fig 3 to Fig 7 below depict the ROC curve for the 5 classifiers.

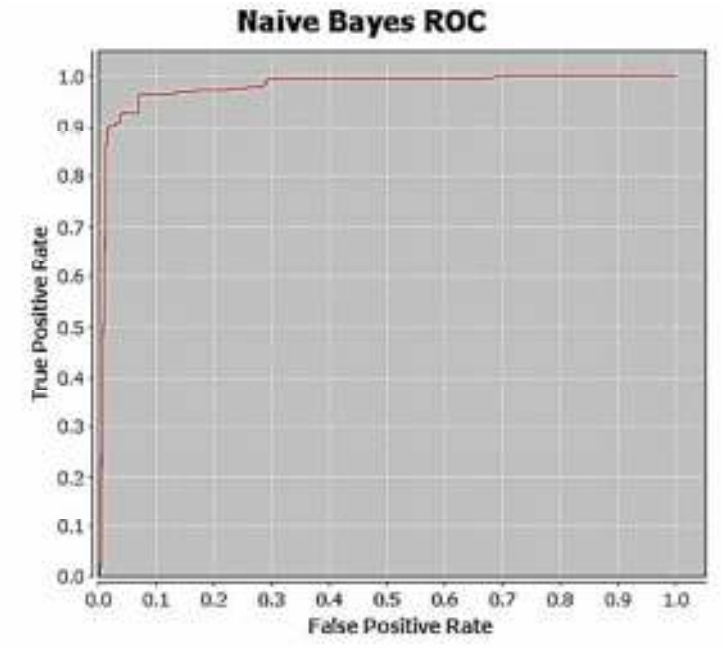

Fig 3. Naïve Bayes ROC Curve

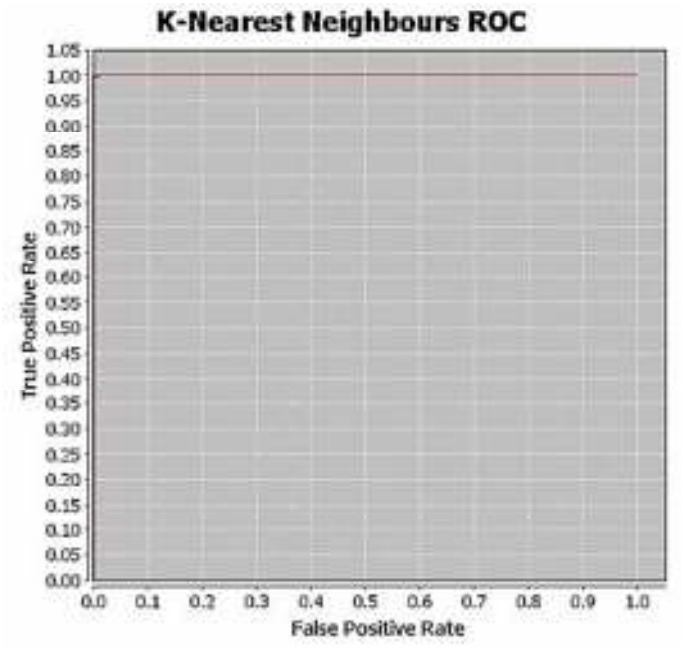

Fig 4. K-nearest Neighbor ROC Curve

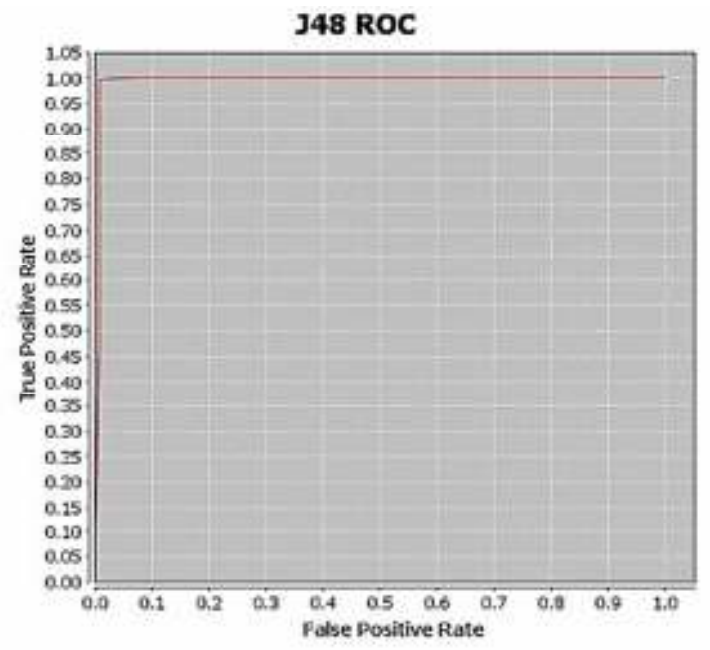

Fig 5. Decision Tree ROC Curve

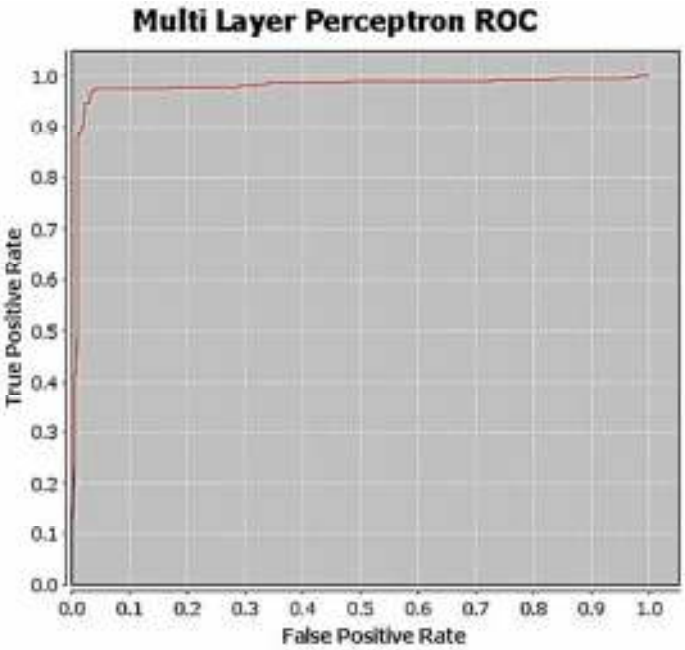

Fig 6. Multi-Layer Perceptron ROC Curve

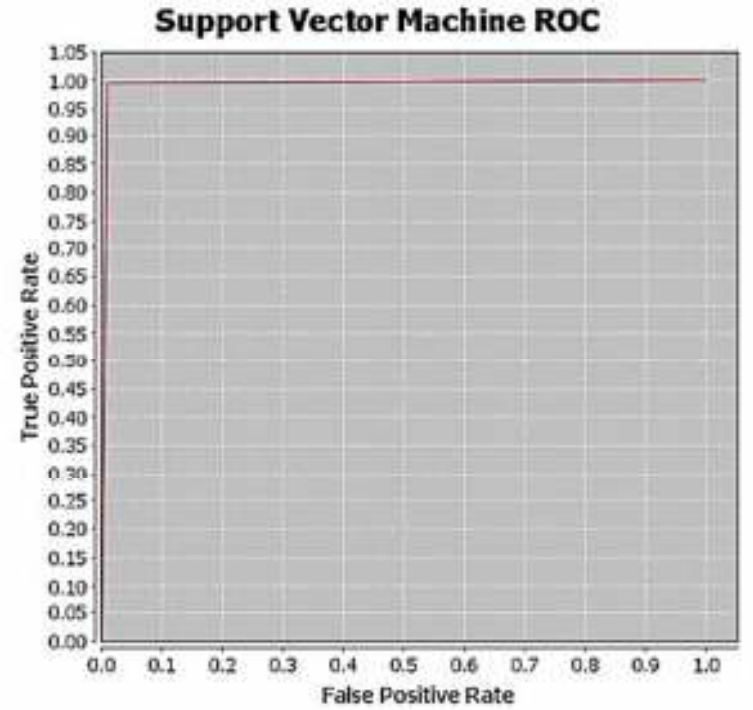

Fig 7. Support Vector Machine ROC Curves

ROC curves signify the tradeoff between false positive and true positive rates, which means that any increase in the true positive rate is accompanied by a decrease in the false positive rate. Then, as shown by the ROC curves, the k-nearest neighbor classifier performed the best result. The line in the k-nearest neighbor diagram is the closest to the left-hand border and the top border compared to other diagrams, indicated that it offers the finest result among the other classifiers.

Furthermore, the area under the curve (AUC) is used to measure the accuracy. An area of 1 means a perfect result while a 0.5 value is a worthless result. The AUC point system is as follows: 0.90 – 1.00 = excellent (A), 0.80 - 0.90 = good (B), 0.70 - 0.80 = fair (C), 0.60 - 0.70 = poor (D) and 0.50 - 0.60 = fail (F). Fig 8 presents the experimental AUC values.

Classifiers	AUC Value
Naïve bayes	0.979
K-nearest neighbor (KNN)	1
Decision Tree (J48)	0.995
MLP	0.979
SVM	0.994

Fig 8. The Values of the Area Under the Curve for ROC Diagrams

It is found that the KNN classifier has the optimum AUC value (1.00) compared to the other classifiers. The decision tree classifier is in second place with 0.995, followed by SVM with 0.994. Naïve Bayes and MLP are fourth and fifth with 0.979 AUC value.

5.0 CONCLUSIONS AND FUTURE WORK

Mobile devices are used by billions of people around the globe. Sensitive data is stored on such devices, from contact lists, to passwords and credit card numbers. The research community is witnessing the manifestation of new mobile malware on a monthly basis, which steals user data and causes many difficulties for the user. In this study, a machine learning approach was used to tackle the security vulnerabilities on mobile devices. Machine learning classifiers were applied to the network traffic of a mobile device to detect malicious activities. Five types of classifiers were used, namely Naïve Bayes, KNN, decision tree, MLP, and SVM. The results are surprisingly high, with a 99.94 % detection rate for KNN.

The data sample employed in this study is one of the newest in the research community. However, with the advent of novel malware each month, it is imperative to collect malware samples continuously and to analyse and improve security systems.

For future work, the presented method may be potentially applied to developing a comprehensive cloud-based intrusion detection system (IDS) using distributed application frameworks in the cloud [31] with the help of virtual machine deployment in the cloud [32]. Alternatively, the same process can be performed on the entire data sample, whereas this study was conducted only on 10% of the sample. Moreover, additional network attributes should be applied in prospective works to inspect network traffic more profoundly.

ACKNOWLEDGMENTS

This work was supported in part by the University of Malaya and the Ministry of Higher Education, Malaysia, under Grant RG086-12ICT and FRGS FP034-2012A.

REFERENCES

[1] Abolfazli S, Sanaei Z, Gani A, Xia F and Yang LT (2013), "Rich Mobile Applications: Genesis, taxonomy, and open issues", *Journal of Network and Computer Applications*, DOI: http://dx.doi.org/10.1016/j.jnca.2013.09.009 [Online]. Available at: http://www.sciencedirect.com/science/article/pii/S1084804513001975 (Accessed: 1st October 2013).

[2] Eslahi M, Salleh R and Anuar NB (2012), "MoBots: A new generation of botnets on mobile devices and networks", *Proceedings of the 2012 IEEE Symposium on Computer Applications and Industrial Electronics (ISCAIE)*, Kota Kinabalu, Malaysia, pp. 262-266.

[3] Teufl P, Ferk M, Fitzek A, Hein D, Kraxberger S and Orthacker C (2013), "Malware detection by applying knowledge discovery processes to application metadata on the Android Market (Google Play)", *Security and Communication Networks*, DOI: 10.1002/sec.675 [Online]. Available at: http://dx.doi.org/10.1002/sec.675 (Accessed: 1st October 2013).

[4] SlideME (2013), "SlideME | Android Apps Market: Download Free & Paid Android Application", Available at: http://slideme.org/ (Accessed: 1st October 2013).

[5] Lookout (2013), "The Bearer of BadNews | The Official Lookout Blog", Available at: https://blog.lookout.com/blog/2013/04/19/the-bearer-of-badnews-malware-google-play/ (Accessed: 1st October 2013).

[6] Anuar NB, Sallehudin H, Gani A and Zakaria O (2008), "Identifying False Alarm for Network Intrusion Detection System Using Hybrid Data Mining and Decision Tree", *Malaysian Journal of Computer Science*, Vol. 21 No. 2, pp. 101-115.

[7] Shamshirband SS, Shirgahi H and Setayeshi S (2010), "Designing of Rescue Multi Agent System Based on Soft Computing Techniques", *Advances in Electrical and Computer Engineering*, Vol. 10 No. 1, pp. 79-83.

[8] Feizollah A, Shamshirband S, Anuar N, Salleh R and Mat Kiah M (2013), "Anomaly Detection Using Cooperative Fuzzy Logic Controller", *Proceedings of the 16th FIRA RoboWorld Congress*, Kuala Lumpur, Malaysia, pp. 220-231.

[9] Garcia-Teodoro P, Diaz-Verdejo J, Maciá-Fernández G and Vázquez E (2009), "Anomaly-based network intrusion detection: Techniques, systems and challenges", *Computers & Security*, Vol. 28 No. 1, pp. 18-28.

[10] Yajin Z and Xuxian J (2012), "Dissecting Android Malware: Characterization and Evolution", *Proceedings of the 2012 IEEE Symposium on Security and Privacy (SP)*, San Fransico, USA, pp. 95-109.

[11] Zheng M, Sun M and Lui J (2013), "DroidAnalytics: A Signature Based Analytic System to Collect, Extract, Analyze and Associate Android Malware", Available at: http://arxiv.org/abs/1302.7212 (Accessed: 1st October 2013).

[12] Sahs J and Khan L (2012), "A Machine Learning Approach to Android Malware Detection", *Proceedings of the 2012 European Intelligence and Security Informatics Conference (EISIC)*, Odense, Denmark, pp. 141-147.

[13] Wu D-J, Mao C-H, Wei T-E, Lee H-M and Wu K-P (2012), "DroidMat: Android Malware Detection through Manifest and API Calls Tracing", *Proceedings of the 2012 Seventh Asia Joint Conference on Information Security (Asia JCIS)*, Tokyo, Japan, pp. 62-69.

[14] Huang CY, Tsai YT and Hsu CH (2013), "Performance Evaluation on Permission-Based Detection for Android Malware", *Proceedings of the International Computer Symposium ICS 2012*, Hualien, Taiwan, pp. 111-120.

[15] Burguera I, Zurutuza U and Nadjm-Tehrani S (2011), "Crowdroid: behavior-based malware detection system for Android", *Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices*, Chicago, Illinois, USA, pp. 15-26.

[16] tPacketCapture-Pro (2013), "tPacketCapture Pro - Android Apps on Google Play", Available at: https://play.google.com/store/apps/details?id=jp.co.taosoftware.android.packetcapturepro (Accessed: 1st October 2013).

[17] Su X, Chuah M and Tan G (2012), "Smartphone Dual Defense Protection Framework: Detecting Malicious Applications in Android Markets", *Proceedings of the 2012 Eighth International Conference on Mobile Ad-hoc and Sensor Networks (MSN)*, Chengdu, China, pp. 153-160.

[18] Anubis (2013), "Anubis: Analyzing Unknown Binaries", Available at: http://anubis.iseclab.org/ (Accessed: 1st October 2013).

[19] CopperDroid (2013), "CopperDroid", Available at: http://copperdroid.isg.rhul.ac.uk/copperdroid/ (Accessed: 1st October 2013).

[20] SandDroid (2013), "SandDroid: An Automatic Android Program Analysis Sandbox", Available at: http://sanddroid.xjtu.edu.cn/ (Accessed: 1st October 2013).

[21] tshark (2013), "tshark - The Wireshark Network Analyzer 1.10.0", Available at: http://www.wireshark.org/docs/man-pages/tshark.html (Accessed: 1st October 2013).

[22] Ghorbani AA, Lu W and Tavallaee M (2010), "*Network intrusion detection and prevention: concepts and techniques",* vol. 47. Springer.

[23] Manocha S and Girolami MA (2007), "An empirical analysis of the probabilistic K-nearest neighbour classifier", *Pattern Recognition Letters*, Vol. 28 No. 13, pp. 1818-1824.

[24] Ripley BD (2008), "*Pattern recognition and neural networks",* Cambridge university press.

[25] Li X-B (2005), "A scalable decision tree system and its application in pattern recognition and intrusion detection", *Decision Support Systems*, Vol. 41 No. 1, pp. 112-130.

[26] Swingler K (1996), "*Applying Neural Networks: A Practical Guide",* illustrated, reprint edition, Morgan Kaufmann.

[27] Wang L (2005), "*Support Vector Machines: theory and applications",* vol. 177. Springer Verlag.

[28] Damopoulos D, Menesidou SA, Kambourakis G, Papadaki M, Clarke N and Gritzalis S (2012), "Evaluation of anomaly-based IDS for mobile devices using machine learning classifiers", *Security and Communication Networks*, Vol. 5 No. 1, pp. 3-14.

[29] Hall M, Frank E, Holmes G, Pfahringer B, Reutemann P and Witten IH (2009), "The WEKA data mining software: an update", *SIGKDD Explor. Newsl.*, Vol. 11 No. 1, pp. 10-18.

[30] Shamshirband S, Anuar NB, Kiah MLM and Patel A (2013), "An appraisal and design of a multi-agent system based cooperative wireless intrusion detection computational intelligence technique", *Engineering Applications of Artificial Intelligence*, Vol. 26 No. 9, pp. 2105–2127.

[31] Shiraz M, Gani A, Khokhar RH and Buyya R (2013), "A Review on Distributed Application Processing Frameworks in Smart Mobile Devices for Mobile Cloud Computing", *Communications Surveys & Tutorials, IEEE*, Vol. 15 No. 3, pp. 1294-1313.

[32] Shiraz M, Abolfazli S, Sanaei Z and Gani A (2013), "A study on virtual machine deployment for application outsourcing in mobile cloud computing", *The Journal of Supercomputing*, Vol. 63 No. 3, pp. 946-964.

Biography

Ali Feizollah received his Bachelor of Information System (IS) from the Ajman University of Science and Technology (AUST), Ajman, UAE in 2010. He started his Master of Computer Science at the University of Malaya, Kuala Lumpur in 2011. His research interests are mobile malware, intrusion detection system.

Nor Badrul Anuar obtained his PhD in Information Security from Centre for Security, Communications and Network Research (CSCAN), Plymouth University, UK in 2012 and Master of Computer Science from University of Malaya, Malaysia in 2003. He is a senior lecturer at the Faculty of Computer Science and Information Technology in University of Malaya, Kuala Lumpur. He has published a number of conference and journal papers locally and internationally. His research interests include information security (i.e. intrusion detection systems), artificial intelligence and library information systems.

Fairuz Amalina received her Bachelor of Engineering Technology (Hons.) in Networking Systems from University of Kuala Lumpur (UniKL) Malaysia. She started her master of Computer Science at the University of Malaya, Kuala Lumpur in 2013. Her research interests are Mobile cloud security and Mobile cloud computing.

Rauf Ridzuan received his B.Eng degree in electronics majoring in computer from Multimedia University Malaysia in 2007. In 2006, he did industrial training at F-Secure and gain exposure with reverse code engineering. Then he continued to work as anti-malware analyst and threat analyst with F-Secure Labs. In July 2013, he moved to Singapore and started working as security response engineer at Symantec with the Symantec's security technology and response team. Throughout 6 years in anti-virus industry, he has experience in reverse code engineering in Windows and Android platform, malware hunting, software development with Python and Perl, and delivering security awareness and basic reverse engineering classes in local universities.

Rosli Bin Salleh received his B.Sc. degree in Computer Science from University of Malaya, Malaysia, in 1994, and the M.Sc and PhD degrees from the University of Salford, United Kingdom in 1997 and 2001 respectively. From 2001, he worked as a lecturer in the Department of Computer System and Technology, Faculty of Computer Science and Information Technology, University of Malaya, Malaysia. He was appointed as a senior lecturer in 2007 and as an Associate Professor in 2013. His research interests include Mobile IPv6 handover and security, Botnet research, and wireless sensor networks.

Shahaboddin Shamshirband received his MSc degree in Computer Science from Islamic Azad University of Mashhad (IAUM), Iran in 2006. He joined the Faculty of Computer Science, Islamic Azad University, Chalous Branch, Iran for seven years. Currently, he is pursuing his PhD from the University of Malaya, Malaysia. His main research covers networking, security, computational intelligence, and cloud computing.