Hindawi Publishing Corporation
Journal of Applied Mathematics
Volume 2014, Article ID 594501, 10 pages
http://dx.doi.org/10.1155/2014/594501

Research Article

Linear SVM-Based Android Malware Detection for Reliable IoT Services

Hyo-Sik Ham,[1] Hwan-Hee Kim,[1] Myung-Sup Kim,[2] and Mi-Jung Choi[1]

[1] *Department of Computer Science, Kangwon National University, 1 Kangwondaehak-gil, Gangwon-do 200-701, Republic of Korea*
[2] *Department of Computer and Information Science, Korea University, 2511 Sejong-ro, Sejong-si 339-770, Republic of Korea*

Correspondence should be addressed to Mi-Jung Choi; mjchoi@kangwon.ac.kr

Received 31 January 2014; Accepted 22 July 2014; Published 3 September 2014

Academic Editor: Young-Sik Jeong

Current many Internet of Things (IoT) services are monitored and controlled through smartphone applications. By combining IoT with smartphones, many convenient IoT services have been provided to users. However, there are adverse underlying effects in such services including invasion of privacy and information leakage. In most cases, mobile devices have become cluttered with important personal user information as various services and contents are provided through them. Accordingly, attackers are expanding the scope of their attacks beyond the existing PC and Internet environment into mobile devices. In this paper, we apply a linear support vector machine (SVM) to detect Android malware and compare the malware detection performance of SVM with that of other machine learning classifiers. Through experimental validation, we show that the SVM outperforms other machine learning classifiers.

1. Introduction

The Internet of Things (IoT) is the communications between things or physical and logical objects organized with networks to extend into a communication network like the existing Internet [1]. It is a generic term of technologies that have intelligent interfaces which actively interact. If things communicate with each other and have intelligent interfaces, they would have new functions beyond their own existing characteristics. The newly obtained properties would bring us convenience and huge usefulness. Machine-to-machine communication or IoT is likely to serve a company with the advancement of smartphones.

IoT technologies and smartphones have been connected to provide a variety of services all over the world. Audi, a German company, offers a service that automatically records data such as mileage and location of electric bicycles through a smartphone [2], while TBWA Helsinki, a company in the Republic of South Africa, provides a service that connects smartphones with a shop window outside a store to check and purchase goods by touching the show window [3]. NEC in Japan installs sensors measuring conditions such as temperature, humidity, and rainfall on a farm to enable smartphones to manage the farmland and crops [4]. Lockitron, an American company, provides a door lock service using smartphones without keys [5]. Likewise, most IoT services are monitored and controlled through smartphone applications.

By combining IoT with smartphones, many convenient IoT services [6] have been provided to users. For example, using smartphone's range of sensors (accelerometer, Gyro, video, proximity, compass, GPS, etc.) and connectivity options (cell, WiFi, Bluetooth, NFC, etc.), we can have a well-equipped Internet of Things device in our pocket that can automatically monitor our movements, location, and workouts throughout the day. The Alohar Mobile Ambient Analytics Platform [7] efficiently collects location and other mobile sensor data and quickly analyzes it to understand a smartphone user's behavior. Through smartphone applications, we can remotely monitor and manage your home and cut down on your monthly bills and resource usage. Smart thermostats like the Nest [8] use sensors, real-time weather forecasts, and the actual activity in your home during the day to reduce your monthly energy usage by up to 30%. We can

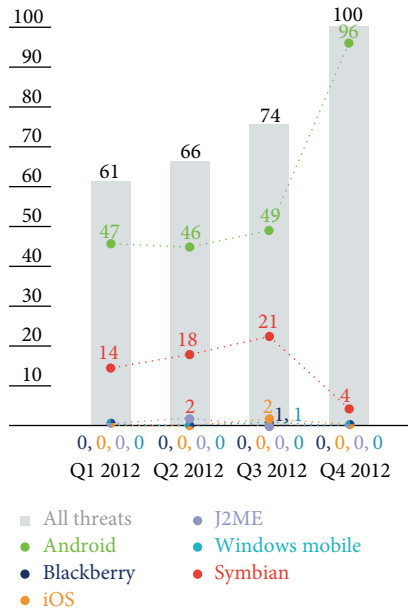

Figure 1: Increase of Android malware.

use the Nest application to connect to the thermostat from a smartphone and can change the temperature miles from home. However, there are adverse underlying effects in this scenario such as invasion of privacy and information leakage. Because there is a diversity of important personal information such as user's location, contact information, and certificates in a smartphone, hackers pose a serious threat [9, 10]. Currently, hackers are expanding their targets from existing PCs to smartphones. Security measures should be prepared to protect users against attacks. A method should be prepared as a security mechanism for detecting and controlling malware that leaks information from smartphones or causes malicious damage through malfunction [11].

Figure 1 is a report by Finnish security company F-secure which states that, 301 mobile malware samples from 2012, 238 samples targeted the Android platform [23]. While the amount of malware that targeted other mobile platforms gradually decreased as time went on from the 1st to 4th quarter, Android showed a contrasting result. The reason for the increase in Android malware was its open source policy and its leniency to market application verification. In addition, it easily allowed the distribution of malware in the market through the repackaging method of inserting it in a normal application.

Previous studies showed various approaches to detecting mobile malware such as signature-based detection [12–15], behavior-based detection [16–20], and taint analysis-based detection [21, 22]. This paper identifies the issues of previous studies and proposes a detection method through a linear support vector machine (SVM) [24] to secure reliable IoT services [25]. The linear SVM shows high performance among machine learning algorithms in order to effectively detect malware in the Android platform with monitored resources during application runtime.

The organization of this paper is as follows. In Section 2, we summarize previous studies on mobile malware detection and briefly introduce the linear SVM algorithm as a related work. In Section 3, we explain resource monitoring information and system for detecting malware. In Section 4, we show experimental results for malware detection using various machine learning classifiers. In Section 5, we conclude this paper and propose possible future work.

2. Related Works

This section examines the trends of previous studies and explains the linear SVM method for detecting mobile malware.

2.1. Mobile Malware Detection Trends. To detect abnormal behaviors occurring in an existing mobile environment (malware, virus, worm, etc.), signature-based detection, behavior-based detection, and taint analysis-based detection were performed. Trends of the studies are summarized in Table 1 based on their detection techniques and collected data.

Signature-based detection [12–15] is a traditional method used to detect malware in a PC environment. To define signature, static and dynamic methods are simultaneously used. Static analysis targets the source and object codes and analyzes the codes without actually starting a program. It decompiles the source code of a malware to discover vulnerabilities that occur in commands, statements, and so on. Dynamic analysis is a method of finding certain patterns in memory leakage, traffic flow, and data flow while actually running the program. However, a large amount of storage is required for applying this method to the mobile environment, and the performance overhead is high for pattern matching.

Behavior-based detection [16–20] is a method of detecting invasion status by comparatively analyzing predetermined attack patterns and process behavior that occur in a system. It is one of the studies that has been receiving the most attention recently due to signature-based detection's limited detection of malicious behavior. To detect abnormal patterns, it mainly monitors event information that occurs in smartphone features such as memory usage, SMS content, and battery consumption. Host-based detection (for directly monitoring information inside a device) and network-based detection (for gathering information via network) are frequently used. Since host-based detection increases the usage of a smartphone's battery and memory, a detection method of collecting data inside the device and transmitting the data to an outside analysis server is mainly used. In addition, a machine learning technique is used to improve the analysis rate of dynamic data. Therefore, it is highly important to choose the proper features to be collected and select a suitable machine learning algorithm for accurate detection.

Dynamic analysis-based detection [21, 22], also called "taint analysis," is a method of marking specific data and monitoring the process of data being sent in an application code to track the flow of data. Since a smartphone runs in a virtual machine, this method is considered appropriate. However, it is no longer being studied due to the difficulty

Table 1: Trends of studies on mobile malware detection techniques.

Detection technique	Author	Collected data	Description
Signature-based technique	Schmidt et al. [12]	Executable file analysis	Uses the readelf command to carry out static analysis on executable files using system calls
	Bläsing et al. [13]	Source code analysis	Uses the Android sandbox to carry out static/dynamic analysis on applications
	Kou and Wen [14]	Packet analysis	Uses functions such as packet-preprocessing and pattern-matching to detect malware
	Bose et al. [15]	API call history	Collects system events of upper layers and monitors their API calls to detect malware
Behavior-based technique	Schmidt et al. [16]	System log data	Detects anomalies in terms of Linux kernels and monitors traffic, kernel system calls, and file system log data by users
	Cheng et al. [17]	SMS, Bluetooth	Lightweight agents operating in smartphones record service activities such as usage of SMS or Bluetooth, comparing the recorded results with users' average values to analyze whether there is intrusion or not.
	Liu et al. [18]	Battery consumption	Monitors abnormal battery consumption of smartphones to detect intrusion by newly created or currently known attacks
	Burguera et al. [19]	System call	Monitors system calls of smartphone kernel to detect external attacks through outsourcing
	Shabtai et al. [20]	Process information	Continuously monitors logs and events and classifies them into normal and abnormal information
Dynamic analysis technique	Fuchs et al. [21]	Data marking	Analyzes malware by carrying out static taint analysis for Java source code
	William et al. [22]	Data marking	Modifies stack frames to add taint tags into local variables and method arguments and traces the propagation process through tags to analyze malware

in applying it in an actual environment and because of the overhead of tracking data flow to a low level.

2.2. Malware Detection via Linear SVM. In this paper, malware is detected based on the collected data by monitoring resources in an Android environment. Behavior-based detection involves the inconvenience of having to determine malware infection status by examining numerous features. Accordingly, behavior-based detection uses a machine learning method to enable automated malware classification and to ensure its identification and accuracy. The machine learning method is a method of entering the data collected from the device as learning data to create a learning model and applying some of the other data to the learning model.

A diversity of classifiers is used for machine learning techniques. Typically, there are DT (decision tree), BN (Bayesian networks), NB (naive Bayesian), Random forest, and SVM (support vector machine). DT [26] is a tree for sorting based on the feature value to classify instances. In this way, it calculates probability values of being able to reach each node and draws a result depending on the probability values. BN [27] is a graphic model that combines a probability theory based on Bayesian theory with a graphic theory. In other words, it makes a conditional probability table with the given data and configures a topology of the graph to draw a conclusion. NB [28] assumes dependent features as independent ones and calculates their probabilities to draw a conclusion. RF [29] combines decision trees formed by the independently sampled random vectors to draw a conclusion and shows a relatively higher detection rate. RF is a machine learning classifier frequently used for malware detection studies in the Android environment [30, 31]. Neural networks technique [32] is another machine learning technique. However, because neural networks technique consumes more time than other classifiers when training [33], it is considered difficult to apply to the malware detection system in which real time is emphasized. Therefore, this paper does not consider neural networks.

In this paper, a linear SVM method [24] is applied to detect malware. SVM is one of the machine learning classifiers receiving the most attention currently, and its various applications are being introduced because of its high performance [34]. The SVM could also solve the problem of classifying nonlinear data. Of the input features, unnecessary ones are removed by the SVM machine learning classifier itself and the modeling is carried out, so there is some overhead in the aspect of time. However, it could be expected to perform better than other machine learning classifiers in the aspect of complexity or accuracy in analysis [35].

Figure 2 shows how to find hyperplanes which are criteria for the SVM to do the learning process to classify data. All hyperplanes (a), (b) and (c) classify two things correctly, but the greatest advantage of the SVM is that it selects hyperplane (c) which maximizes the margin (the distance between data) and accordingly maximizes the capability of generalization. Therefore, even if input data is located near a hyperplane, it has an advantage of being able to classify more correctly compared to other classifiers. We verify that

Table 2: Selected features for malware detection.

Resource type		Resource feature
Network		RxBytes, TxBytes, RxPacket, TxPacket
Telephone		Send/receive call
SMS Message		Send/receive SMS
CPU		CPU usage
Battery		Level, temperature, voltage
Process		Process ID, process name, running process, context switches
Memory	Native	Total size, shared size, allocated size, physical page, virtual set size, free size, heap size, dirty page
	Dalvik	Total size, shared size, allocated size, physical page, virtual set size, free size, heap size, dirty page

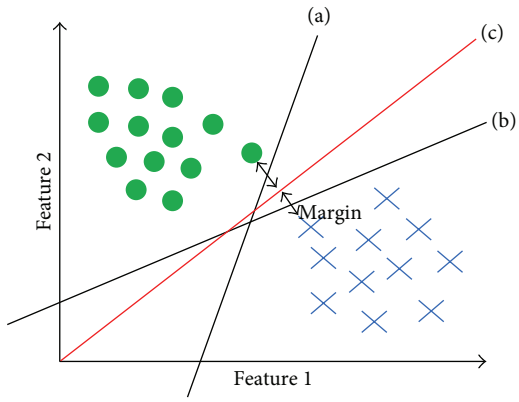

Figure 2: Data classification method of SVM.

SVM shows a good detection performance by comparing the experimental results of SVM with those of other machine learning classifiers (Bayesian network, decision tree, naïve Bayesian, and random forest) and SVM analysis technique.

3. Collection of Resource Information for Malware Detection

This section presents a method of collecting resource information for detecting Android malware. It explains collected resource features and agents designed and implemented to collect resource information inside Android devices.

3.1. Resource Features for Malware Detection. For detecting malware that is the target of analysis, resource information generated in a device is monitored when a user executes normal applications or abnormal applications infected with malware. In a previous study [20], every resource and event generated in an Android device was defined and all these features were used for analyzing malware.

However, the number of features is 88, which are too many, most of them having low correlation, with the Android memory structure not being reflected. In addition, some of these 88 features could be extracted only if the root permission is acquired. The 32 features proposed in this paper are information that could be extracted even without the root permission. In this paper, 32 features that are highly related to targeted malware, as shown in Table 2, are defined by classifying them into seven categories according to resource type. This study does not monitor the total memory usage that simply changes through an application execution but monitors the usage amount classified into native area and Dalvik machine area by considering the memory characteristics of the Android platform. Dalvik machine memory is allocated when running each application.

For the features proposed in this paper, every feature was extracted about network, phone, message, CPU, battery, and memory for each process. The existing study [20] used a feature selection algorithm such as the information gain to increase the detection system's performance, but this paper did not carry out the feature selection. As also mentioned in Section 2.2, the reason was because the SVM classifier autonomously carried out dimensional reduction function to use only the required features for determining results.

3.2. Malware Detection System Architecture. To monitor the selected resource features, an agent is needed that can continuously monitor the corresponding features inside a device. This experiment alternatively executes a normal application and an abnormal application on the Android platform to test malware detection. Figure 3 shows the structure of the Android malware detection system, which primarily consists of a mobile agent and an analysis server.

First, the mobile agent collects information for each application through the resource monitoring component. The data is collected from the Linux kernel in the mobile agent, and the feature extractor is responsible for the collecting of actual data. The feature extractor is comprised of four collectors, and they collect information on variations in network, memory, CPU, and battery. The collected feature information is specified in Table 2 of Section 3.1. This collected information is transferred to the data management module, and the data management module transforms the collected information into a vector form. The data constructed as a vector form by the data management module is transferred to the analysis server for evaluation. At such time, the reason for transmitting data to an external server is because its overhead is large in the aspect of time and resource if the modeling and analysis are carried out by machine learning in the mobile device. Therefore, to minimize such an overhead, malware detection is carried out in the analysis server, and only the detection result is transferred again to the mobile agent.

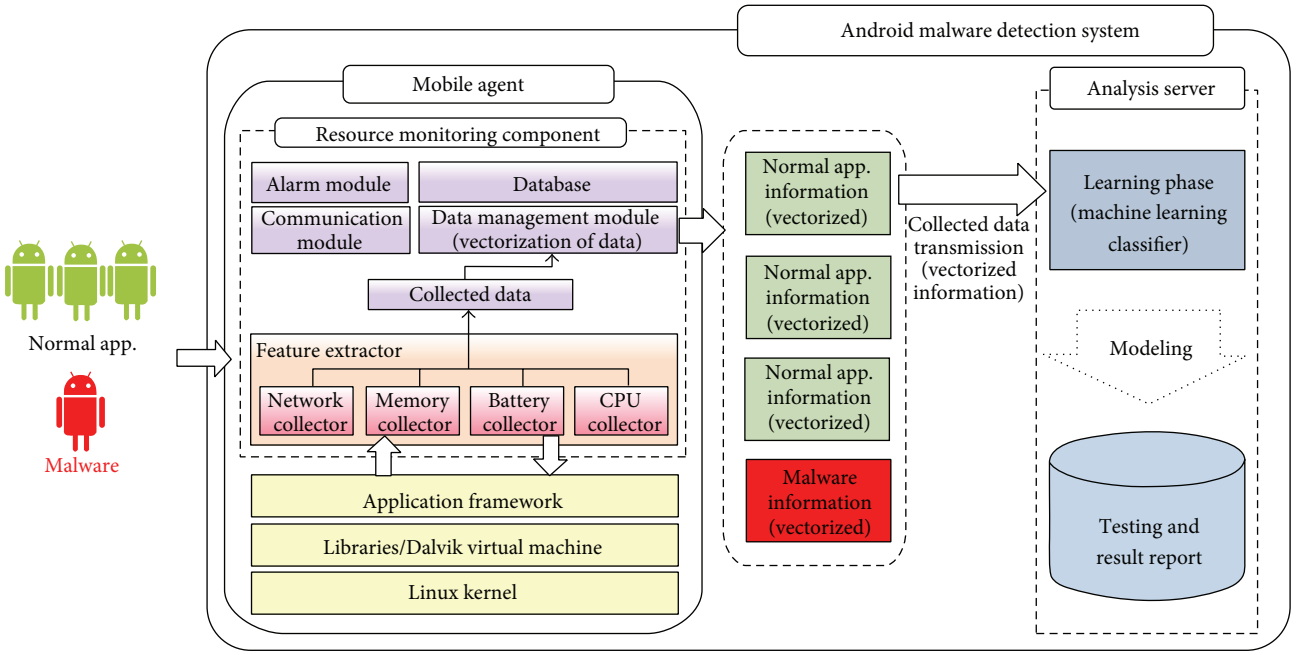

Figure 3: Android malware detection system architecture.

The analysis server learns by using the vectorized resource data for each application, which is transferred from the mobile agent as input data. After learning, a model (pattern) of the resource data for each application is created and, based on it, the existence of malware is determined. If malware is detected, an alarm message is transmitted to the user through the alarm module. Figure 4 represents the algorithm of the malware detection system proposed in this paper as a sequence diagram.

Examining the overall flow of the algorithm, it extracts feature information for each application and makes the machine learning classifier to learn the extracted information. Based on this learned information, it determines the existence of malware. This method is not much different from existing malware detection studies. Upon comparing this paper with the existing studies, however, a difference is found on the information on features and the applied machine learning classifier.

4. Experimental Results

This section applies the proposed linear SVM technique. It demonstrates the superiority of the SVM by comparing it with four machine learning classifiers and describes the experimental methods and results.

4.1. Android Malware Categories. This study chooses 14 of the latest malware programs for each category to verify the proposed method. Malicious applications are selected on the basis of the "typical cases of malware causing great damage to users" presented in the 2012 ASEC report [36] from Ahnlab in Korea. Most of the Android-targeted malware is divided into Trojan, spyware, root permission acquisition (exploit), and installer (dropper). The reason for Trojan having a large proportion of the selected malware is because most of the malicious codes that occurred in 2012 were Trojan. Table 3 describes the malware to be analyzed in this study.

(i) Trojan: it looks harmless, but it is a program containing a risk factor in effect. Malware is usually included in the program, so it basically executes the malware when running the application.

(ii) Spyware: a compound word formed from "spy" and "software" and it is a type of malware that is secretly installed on a device to collect information. It is frequently used for commercial uses such as repeatedly opening pop-up advertisement or redirecting users to a particular website. Spyware causes inconvenience by changing a device's settings or being difficult to delete.

(iii) Root permission acquisition (exploit): it uses unknown vulnerabilities or 0-day attacks. The new vulnerability is discovered but not yet patched for. It is malware that acquires root permission to clear security settings and makes additional attacks on the Android platform.

(iv) Installer (dropper): it conceals malware in a program and guides users to run malware and spyware. These days, because it does not install one kind of malware but multiple ones with the advent of multidroppers, it makes detection more difficult.

4.2. Elements of Data Set. This paper uses 14 normal applications and 14 malicious ones embedded with malware to test malware detection. The data set is composed of 90% normal and 10% malicious applications. The reason for composing

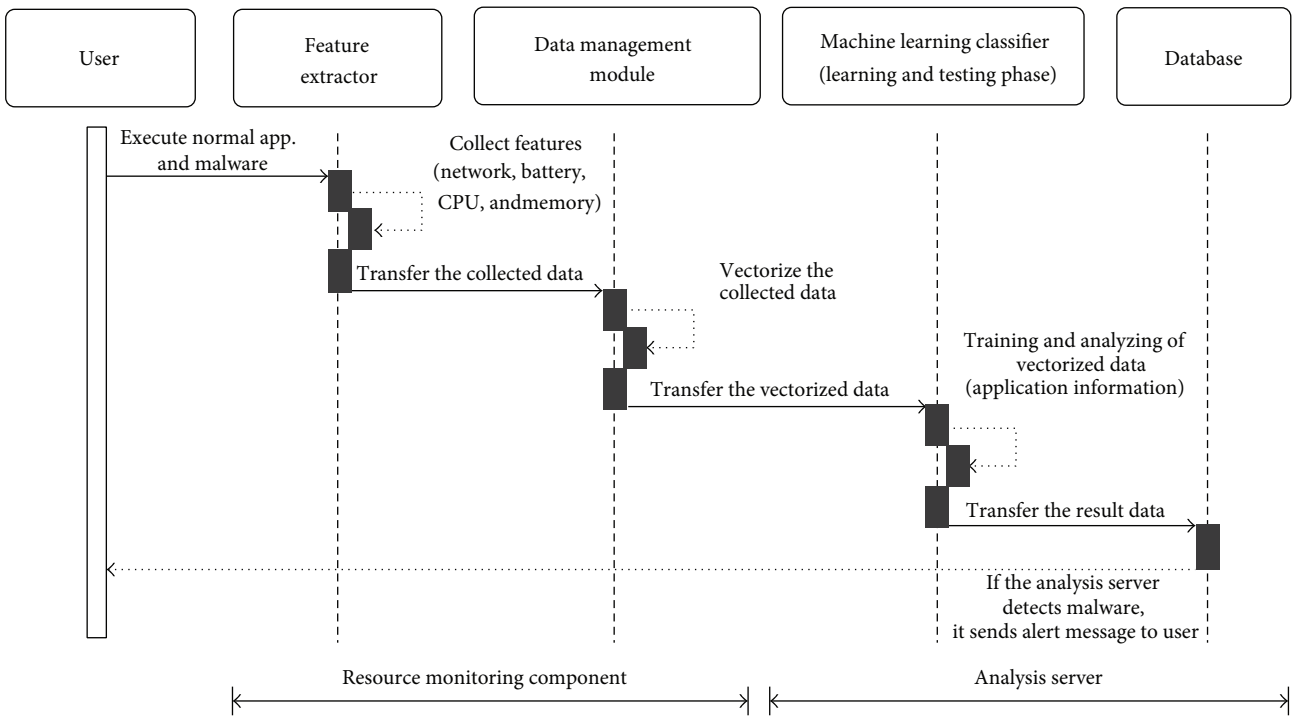

Figure 4: Sequence diagram for malware detection system.

Table 3: Features of malware to be analyzed.

Malware category	Malware Name	Features
Trojan	Zitmo	Disguises as an Android security application
	DroidKungFu	Leaks personal information
	Opfake	Disguises as a game application (performance degradation)
	FakeInst	Disguises as a game application (performance degradation)
	Goldream	Disguises as a game/animation application
	LightDD	Disguise as an adult application
Spyware	Geimini	Carries out a backdoor function
	Adrd.AQ	Carries out a backdoor function
	Snake	Disguises as a game to leak information
	Pjapps	Adds malicious functions to a normal app.
Root permission acquisition (exploit)	Rootor.BT	Makes terminal rooting (security dismantling)
	Basebridge	Acquires root permissions and then communicates with an external server
Installer (dropper)	SMSHider	Guides to install malware through SMS
	Anserver	Downloads other malware

the data set in this way is that normal applications are more common than malicious ones when examining the ratio of applications used in the actual mobile environment. In experiment, we construct the data set using a 5-fold cross-validation method.

Figure 5 shows the 5-fold cross-validation method applied to the data collected from respective devices. As shown in Figure 5, the data collected from other devices are crossed to organize the training and test sets. If the dataset is organized like this, all the collected data are organized as the training and test sets, so it could be said that it is a method considering portability between devices. In other words, it shows that malware detection is possible even if the device's environment is different. It could also be verified that the selected features are useful for detecting malware.

4.3. Evaluation Indicators. This section describes evaluation indicators to verify the performance of experimental results. The indicators used in this paper are TPR (true positive rate), FPR (false positive rate), precision, accuracy, and *F*-measure. Statistical information for the decision result is

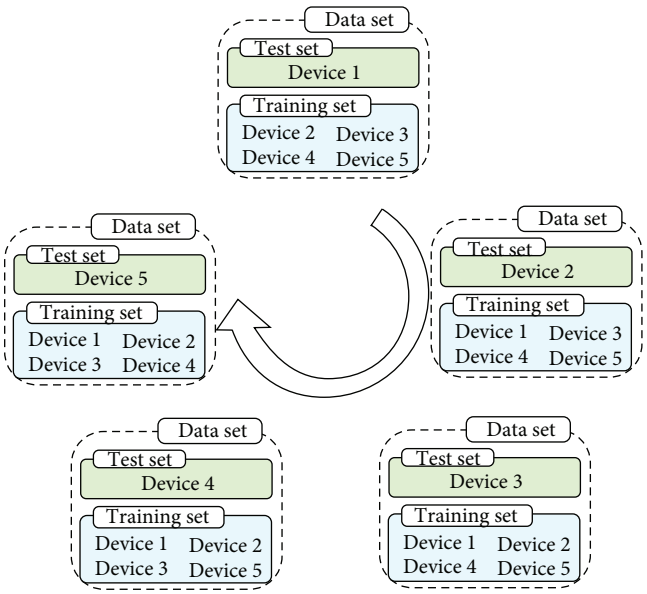

Figure 5: Composition of training and test data set.

Table 4: Confusion matrix of evaluation indicators.

		Predicted data	
		Positive	Negative
Actual data	Positive	TP (true positive)	FN (false negative)
	Negative	FP (false positive)	TN (true negative)

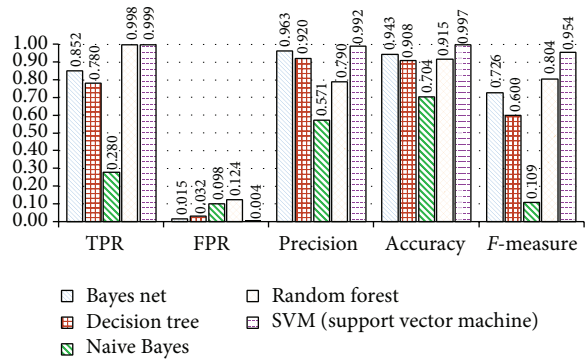

Figure 6: Detection results of respective classifiers.

required to find the respective evaluation indicators. Table 4 is a confusion matrix for computing the evaluation indicators.

TP (true positive) is a numerical value of identifying the uninfected status of a normal application. TN (true negative) represents a number that correctly identifies an application containing malware. FN (false negative) means a number that incorrectly finds malware in an actually normal application. FP (false positive) represents a number that incorrectly finds no malware despite an application actually containing malware. Based on the statistical information above, this paper finds TPR (true positive rate), FPR (false positive rate), precision, accuracy, and F-measure. Equations (1)–(5) for respective indicators are as follows:

$$\text{TPR} = \frac{\text{TP}}{\text{TP} + \text{TN}}, \tag{1}$$

$$\text{FPR} = \frac{\text{FP}}{\text{FP} + \text{TN}}, \tag{2}$$

$$\text{Precision} = \frac{\text{TP}}{\text{FP} + \text{FP}}, \tag{3}$$

$$\text{Accuracy} = \frac{\text{TP}}{\text{TP} + \text{FP}}, \tag{4}$$

$$F\text{-measure} = 2 * \frac{(\text{Precision} * \text{Recall})}{(\text{Precision} + \text{Recall})}. \tag{5}$$

True positive rate (TPR) represents the proportion (1) of correctly identified normal applications. False positive rate (FPR) represents the proportion (2) of malware-containing applications incorrectly identified as safe. If applications containing malware are misdiagnosed, they could cause serious damage to the system, so this indicator is considered important. Precision is an indicator representing an error of the decision value, which represents the proportion (3) of correctly diagnosed normal applications. Accuracy is an indicator representing the system's accuracy, expressed in the proportion (4) of correctly identified normal applications and ones containing malware, respectively, among the results. F-measure is also called $F1$-score and means accuracy (5) in the aspect of decision results.

4.4. Experimental Results. Figure 6 shows malware detection results according to machine learning classifiers. From the TPR perspective, the random forest (TPR = 0.998) and SVM (TPR = 0.999) show a good performance. For the FPR used as the most important evaluation indicator when detecting malware, SVM has FPR = 0.004, which could be determined as the best classifier because its ratio of incorrectly classifying normal applications as malicious is small, and it shows far better performance than other classifiers also in terms of accuracy and precision.

Table 5 shows the results of the detailed malware detection of respective classifiers' TPR/FPR indicators. RF has Adrd.AQ (TPR = 1.000), Anserver (TPR = 0.996), and Geimini (TPR = 0.962), which show higher performance than other classifiers. For other malware, however, it is shown that SVM gives higher performance with TPR = 0.953 on average. In particular, NB does not at all detect specific malware (Adrd.AQ, Anserver, DroidKungFu, GoldDream, Opfake, PjApps, SMSHider, and Snake). For Opfake, SVM gives relatively lower performance with TPR = 0.820. The reason is that Opfake is expanded from FakeInst, which shows similar patterns, so it incorrectly detects Opfake as FakeInst. However, it shows that TPR is about 31% more improved than the random forest. Every classifier shows a low numerical value in terms of FPR, but upon analysis of the correlation with TPR it could be found that SVM shows the best performance. Because the NB classifier's TPR is also 0.000 if its FPR is 0.000, it could be said that NB is a classifier unsuitable for detecting malware.

Table 5: Detailed performance indicators of machine learning classifiers (TPR/FPR).

Normal and malware	TPR					FPR				
	BN	DT	NB	RF	SVM	BN	DT	NB	RF	SVM
Normal	0.852	0.780	0.280	0.998	**0.999**	0.015	0.032	0.098	0.124	**0.004**
Adrd.AQ	0.695	0.671	0.000	**1.000**	0.957	0.012	0.017	**0.000**	0.004	0.002
Anserver	0.985	0.918	0.000	**0.996**	0.957	0.051	0.117	**0.000**	0.004	**0.000**
Basebridge	0.692	0.862	0.487	0.671	**0.939**	0.009	0.056	0.081	0.014	**0.000**
DroidKungFu	0.720	0.868	0.000	0.874	**0.977**	0.008	**0.000**	**0.000**	**0.000**	0.001
FakeInst	0.946	0.709	0.263	0.838	**0.985**	0.005	**0.000**	0.001	0.001	0.011
Geimini	0.649	0.464	0.000	**0.962**	0.893	0.004	0.009	**0.000**	**0.000**	0.001
GoldDream	0.567	0.298	0.000	0.717	**0.994**	0.012	0.005	**0.000**	0.022	0.002
LightDD	0.663	0.562	0.373	0.645	**0.957**	0.012	0.035	0.284	**0.000**	**0.000**
Opfake	0.567	0.429	0.000	0.509	**0.820**	0.005	0.002	**0.000**	0.001	0.005
PjApps	0.946	0.659	0.000	0.548	**0.996**	0.032	0.012	**0.000**	0.003	0.003
RooterBT	0.868	0.451	0.782	0.573	**0.966**	0.009	**0.000**	0.318	0.008	0.004
SMSHider	0.778	0.766	0.000	0.773	**0.949**	0.001	0.054	**0.000**	0.001	0.001
Snake	0.422	0.205	0.000	0.703	**0.935**	0.013	0.007	**0.000**	0.001	0.001
Zitmo	0.750	0.503	0.378	0.789	**0.967**	0.060	0.025	0.087	0.033	**0.001**
Average	0.740	0.610	0.171	0.773	**0.953**	0.017	0.025	0.058	0.014	**0.002**

Table 6: Detailed performance indicators of machine learning classifiers (precision/accuracy/F-measure).

Normal and malware	Precision					Accuracy					F-measure				
	BN	DT	NB	RF	SVM	BN	DT	NB	RF	SVM	BN	DT	NB	RF	SVM
Normal	0.963	0.920	0.571	0.790	**0.992**	0.943	0.908	0.704	0.915	**0.997**	0.904	0.844	0.375	0.882	**0.995**
Adrd.AQ	0.682	0.590	0.000	0.893	**0.939**	0.978	0.972	0.964	**0.996**	**0.996**	0.689	0.628	0.000	0.943	**0.948**
Anserver	0.532	0.315	0.000	0.933	**0.993**	0.951	0.885	0.945	0.996	**0.997**	0.691	0.469	0.000	0.963	**0.975**
Basebridge	0.803	0.455	0.246	0.724	**0.999**	0.976	0.940	0.897	0.970	**0.997**	0.744	0.596	0.327	0.696	**0.968**
DroidKungFu	0.842	**1.000**	0.000	0.997	0.983	0.977	0.993	0.945	0.993	**0.998**	0.776	0.929	0.000	0.932	**0.980**
FakeInst	0.910	**1.000**	0.911	0.973	0.836	**0.992**	0.985	0.960	0.990	0.989	**0.928**	0.830	0.408	0.900	0.905
Geimini	0.842	0.607	0.000	**0.996**	0.957	0.986	0.976	0.971	**0.999**	0.995	0.733	0.526	0.000	0.979	**0.924**
GoldDream	0.730	0.780	0.000	0.653	**0.962**	0.964	0.956	0.945	0.963	**0.997**	0.639	0.431	0.000	0.683	**0.978**
LightDD	0.765	0.481	0.070	0.997	**0.998**	0.971	0.943	0.697	0.981	**0.998**	0.710	0.518	0.118	0.783	**0.977**
Opfake	0.878	0.910	0.000	**0.979**	0.900	0.972	0.966	0.945	0.972	**0.985**	0.689	0.583	0.000	0.670	**0.858**
PjApps	0.554	0.707	0.000	0.880	**0.941**	0.967	0.975	0.959	0.978	**0.997**	0.699	0.682	0.000	0.675	**0.967**
RooterBT	0.846	**1.000**	0.117	0.802	0.926	0.985	0.972	0.687	0.971	**0.994**	0.857	0.621	0.203	0.669	**0.946**
SMSHider	**0.983**	0.451	0.000	0.972	0.976	0.987	0.936	0.946	0.986	**0.996**	0.868	0.568	0.000	0.861	**0.962**
Snake	0.651	0.646	0.000	**0.987**	0.977	0.955	0.949	0.945	0.983	**0.995**	0.512	0.312	0.000	0.821	**0.956**
Zitmo	0.325	0.439	0.144	0.479	**0.977**	0.933	0.958	0.893	0.960	**0.998**	0.454	0.469	0.209	0.596	**0.972**
Average	0.754	0.687	0.137	0.870	**0.957**	0.969	0.954	0.894	0.977	**0.995**	0.726	0.600	0.109	0.804	**0.954**

Table 6 shows the detailed results of respective classifiers' precision/accuracy. For the decision tree, the precision of DroidKungFu, FakeInst, and RooterBT is 1.000, which marks the best performance. Their average precision is 0.687, which is lower than SVM (precision = 0.957). For accuracy, SVM shows higher performance with 0.995 on average. For the F-measure, it is found that the SVM is 0.954 on average except for FakeInst, which gives superior performance from other classifiers.

5. Conclusion and Future Work

This paper proposed an Android malware-detection mechanism using machine learning algorithms for reliable IoT services. This paper also proposed a machine learning technique to remedy the disadvantage of the behavior-based technique (one of the mobile detection techniques) and to correctly detect malware targeting the Android platform. The first problem of existing studies was that they were not suitable

for generalization because they were not able to analyze many types of malware. To solve this problem, the recent domestic trend of malware targeting Androids was evaluated and 14 malware programs were selected to apply them to the proposed method. Second, because the features of the existing papers focused only on the detection of some types of malware or they had no correlation with malware, their detection rate was reduced. This paper reflected the structural characteristics of the Android platform to subdivide its memory space. This study also selected the features having much correlation with malware to increase efficiency. Third, the portability between devices was considered to verify it through the 5-fold cross-validation experimental method. We concluded that the SVM technique could accurately detect most malware in a relative sense by comparatively analyzing them with four classifiers (Bayesian network, decision tree, naïve Bayesian, and random forest).

Future studies may consider exposing hardly detectable malware by resource information and sharper system accuracy. Because diverse variants and new types of mobile malware are on the rise, further study on a technique that could detect future malware should be scheduled. We plan to develop an efficient and lightweight implementation of the SVM algorithm that can be embedded to a smartphone for real-time detection. We also plan to conduct malware elimination and control by applying detection results to actual mobile devices and networks.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgment

This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Science, ICT & Future Planning (2013R1A1A3011698).

References

[1] N. Gershenfeld, R. Krikorian, and D. Cohen, "The internet of things," *Scientific American*, vol. 291, no. 4, pp. 76–81, 2004.

[2] Readwrite, "The Future of Connected Cars: What Audi is Driving Towards," 2012, http://readwrite.com.

[3] Brandingmagazine, *Adidas' Interactive Window Shopping Experience*, 2012, http://www.brandingmagazine.com/.

[4] K. Moessner, F. Le Gall, P. Cousin et al., "Internet of things strategic research and innovation agenda," in *Internet of Things: Converging Technologies for Smart Environments and Integrated Ecosystems*, 2013.

[5] J. Jensen, S. Copenhagen, and A. H. Larsen, *Smart Intercom-Enhancing the Apartment Intercom System*, ACM Computing Classification System, 2012.

[6] T.-T. Truong, M.-T. Tran, and A.-D. Duong, "Improvement of the more efficient & secure ID-based remote mutual authentication with key agreement scheme for mobile devices on ECC," *Journal of Convergence*, vol. 3, no. 2, pp. 25–36, 2012.

[7] Aloha, "Mobile Ambient Analytics Platform," https://www.alohar.com/developer/.

[8] Nest, "Nest Thermostat," https://nest.com/.

[9] D. Werth, A. Emrich, and A. Chapko, "An ecosystem for user-generated mobile service," *Journal of Convergence*, vol. 3, no. 4, 2012.

[10] J. W. K. Gnanaraj, K. Ezra, and E. B. Rajsingh, "Smart card based time efficient authentication scheme for global grid computing," *Human-centric Computing and Information Sciences*, vol. 3, article 16, 2013.

[11] K. Sakurai and K. Fukushima, "Actual condition and issues for mobile security system," *Journal of Information Processing Systems*, vol. 3, no. 2, pp. 54–63, 2007.

[12] A. Schmidt, A. Camtepe, and S. Albayrak, "Static smartphone malware detection," in *Proceedings of the 5th Security Research Conference (Future Security 2010)*, p. 146, 2010.

[13] T. Bläsing, L. Batyuk, A.-D. Schmidt, S. A. Camtepe, and S. Albayrak, "An android application sandbox system for suspicious software detection," in *Proceedings of the 5th International Conference on Malicious and Unwanted Software (Malware '10)*, pp. 55–62, Nancy, France, October 2010.

[14] X. Kou and Q. Wen, "Intrusion detection model based on android," in *Proceedings of the 4th IEEE International Conference on Broadband Network and Multimedia Technology (IC-BNMT '11)*, pp. 624–628, October 2011.

[15] A. Bose, X. Hu, K. G. Shin, and T. Park, "Behavioral detection of malware on mobile handsets," in *Proceedings of the 6th International Conference on Mobile Systems, Applications, and Services*, pp. 225–238, June 2008.

[16] A.-D. Schmidt, H.-G. Schmidt, J. Clausen et al., "Enhancing security of linux-based android devices," in *Proceedings of the 15th International Linux Kongress*, Lehmann, October 2008.

[17] J. Cheng, S. H. Y. Wong, H. Yang, and S. Lu, "SmartSiren: virus detection and alert for smartphones," in *Proceedings of the 5th International Conference on Mobile Systems, Applications and Services (MobiSys '07)*, pp. 258–271, June 2007.

[18] L. Liu, G. Yan, X. Zhang, and S. Chen, "VirusMeter preventing your cellphone from spies," in *Recent Advances in Intrusion Detection* , vol. 5758 of *Lecture Notes in Computer Science*, pp. 244–264, 2009.

[19] I. Burguera, U. Zurutuza, and S. Nadjm-Tehrani, "Crowdroid behavior-based malware detection system," in *Proceedings of the 1st ACM Workshop on Security and Privacy in Smartphones and Mobile Devices (SPSM '11)*, 2011.

[20] A. Shabtai, U. Kanonov, Y. Elovici, C. Glezer, and Y. Weiss, ""Andromaly": a behavioral malware detection framework for android devices," *Journal of Intelligent Information Systems*, vol. 38, no. 1, pp. 161–190, 2012.

[21] A. P. Fuchs, A. Chaudhuri, and J. S. Foster, "Scan-Droid Automated Security Certification of Android Applications," 2011.

[22] E. William, P. Gilbert, C. Byung-Gon et al., "TaintDroid: an information-flow tracking system for realtime privacy monitoring on smartphones," in *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation (OSDI '10)*, pp. 1–6, USENIX Association, Berkeley, Calif, USA.

[23] F-Secure, "Mobile Threat Report," Q4, 2012.

[24] C. J. C. Burgesm, "A tutorial on support vector machines for pattern recognition," *Data Mining and Knowledge Discovery*, vol. 2, no. 2, pp. 121–167, 1998.

[25] C. M. Medaglia and A. Serbanati, "An overview of privacy and security issues in the internet of things," in *The Internet of Things*, pp. 389–395, Springer, New York, NY, USA, 2010.

[26] S. B. Kotsiantis, "Supervised machine learning: a review of classification techniques," *Informatica*, vol. 31, no. 3, pp. 249–268, 2007.

[27] N. Friedman, D. Geiger, and M. Goldszmidt, "Bayesian network classifiers," *Machine Learning*, vol. 29, no. 2-3, pp. 131–163, 1997.

[28] R. Kohavi, *Scaling up the Accuracy of Naive-Bayes Classifiers: A Decision-Tree Hybrid*, KDD, 1996.

[29] L. Breiman, "Random forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001.

[30] H. S. Ham and M. J. Choi, "Analysis of Android malware detection performance using machine learning classifiers," in *Proceedings of the International Conference on ICT Convergence (ICTC '13)*, pp. 490–495, 2013.

[31] T. Kim, Y. Choi, S. Han et al., "Monitoring and detecting abnormal behavior in mobile cloud infrastructure," in *Proceedings of the IEEE Network Operations and Management Symposium (NOMS '12)*, pp. 1303–1310, April 2012.

[32] M. T. Hagan, H. B. Demuth, and M. H. Beale, *Neural Network Design*, vol. 1, Pws, Boston, Mass, USA, 1996.

[33] S. Mukkamala, G. Janoski, and A. Sung, "Intrusion detection using neural networks and support vector machines," in *Proceedings of the International Joint Conference on Neural Networks (IJCNN '02)*, vol. 2, pp. 1702–1707, IEEE, May 2002.

[34] Y. Hwang, J. Kwon, J. Moon, and S. Cho, "Classifying malicious web pages by using an adaptive support vector machine," *Journal of Information Processing Systems*, vol. 9, no. 3, pp. 39–404, 2013.

[35] J. A. K. Suykens and J. Vandewalle, "Least squares support vector machine classifiers," *Neural Processing Letters*, vol. 9, no. 3, pp. 293–300, 1999.

[36] Ahnlab, "Ahnlab ASEC Report," 2012.