Linux Kernel-based Feature Selection for Android Malware Detection

Hwan-Hee Kim
Dept. of Computer Science
Kangwon National University
Chuncheon, Korea
hwanhee0920@kangwon.ac.kr

Mi-Jung Choi*
Dept. of Computer Science
Kangwon National University
Chuncheon, Korea
mjchoi@kangwon.ac.kr

***Abstract*— As usage of mobile increased, target of attackers has changed from PC to Mobile environment. In particular, various attacks have occurred in android platform because it has feature of open platform. To solve this problem, researches of machine learning-based malware detection continually have progressed. However, as version of Android platform continuously is updated, some feature that used in existing research could not collect any more. Therefore, we propose Linux kernel-based novel feature in order to detect malware in higher than android version 4.0. In addition, we perform feature selection to select optimal feature about foregoing feature. This way is able to improve performance of malware detection system. In experiment, by performing android malware detection through support vector machine classifier which has showed relatively good performance in existing studies, we show novel feature feasibility and validity.**

Keywords—Feature selection, Malware detection, Linux kernel, Machine learning

I. INTRODUCTION

In parallel with the increase of mobile applications, mobile devices continue to evolve for providing a variety of services to users [1]. As users' demand increases, the computing power of mobile environment is developing as much as the existing PC environments. Making frequent use of mobile devices started to store various personal information such as bank information and individual's ID, password for each site in the mobile devices. Consequently, attackers tend to widen their attack scope to mobile environments as well as the existing PC ones [2].

Of mobile platforms, various attacks have been carried out frequently toward the Android platform in particular because it has a feature of open platform. In other words, because its verification procedure for third-party applications developed by users is a little bit weaker than other platforms, it is easy to distribute, and capable of repackaging to add some malicious codes into the existing application to re-upload, so it means that the distribution of malware could be easy to carry out. According to the F-secure report [3], it could be shown that the proportion of Android platform gradually increases but the one of other platforms decreases in regard to the generation of new malware. Fig. 1 represents the generation of new malware for each platform described in the F-secure report.

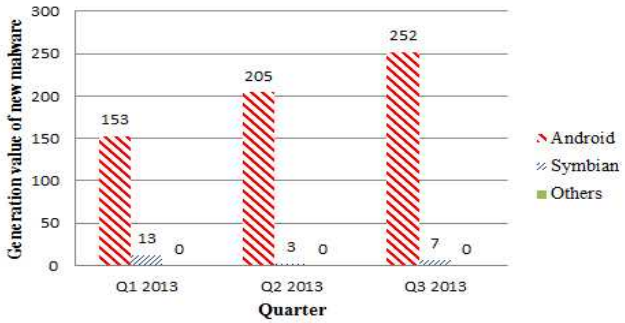

Fig. 1. Generation value of platform-specific new malware

Diverse studies have been conducting actively to cope with these growing malware [4, 5, 6, 7, 8, 9]. Most of these studies could be divided into signature-, behavior- and statistics-based analysis methods. The signature-based analysis method provides a higher detection rate when detecting malware existed in its signature, but has a limit of being non-detectable when finding a new malicious code not existed in its signature. Therefore, the behavior-based analysis method has been exploited recently in a lot of studies to solve the limit of signature-based analysis, and most of them are studies to detect mobile malware based on machine learning [6, 7, 8, 9].

The machine learning based malware detection studies generally extract specific features from mobile devices to send them to an analysis server [10]. The analysis server receiving the feature information carries out its analysis through internal machine learning, and performs malware detection by resending the result. However, there was a problem of not extracting some of the features suggested by the existing studies because the version of Android recently continues to upgrade. In particular, as the version of Android was upgraded from 2.3 to above 4.0, some of the feature information suggested by the existing studies disappeared, or were changed into features being extractable only if acquiring the root authority.

Therefore, this paper proposes new Linux kernel based features being extractable from the Android with a version

* Correspondence to Mi-Jung Choi, Dept. of Computer Science, KNU, Chuncheon, Republic of Korea.

above 4.0 without the root authority. In addition, it carries out feature selection which is a method to improve the detection system's performance by removing unnecessary information from the extracted feature information. These selected features are used to carry out a malware detection experiment in the Android environment. The SVM (support vector machine), which showed relatively better performance in past studies, is exploited for a machine learning classifier used in the experiment.

II. RELATED WORK

The paper in [6] proposed a behavior based malware detection framework in the Android environment. The method proposed in [6] was to classify and detect normal applications and malicious ones displaying abnormal behavior by continuously monitoring various features in mobile devices and applying the extracted features to the machine learning based detection system. Most of the features used in the paper above could be classified into 14 such as keyboard, scheduler, CPU, SMS message, battery, memory and network, and it used a total of 88 features. In addition, it also carried out a feature selection to select the optimal feature for improving the detection system's performance. However, as the Android's version was upgraded, it could not extract features about memory usage for each process such as Garbage_Collections and DMA_Allocation without the root authority. Therefore, there is a limit of being difficult to use in the present Android devices.

The paper in [7] also proposed a learning machine based malware detection system in the Android environment. Most of the features used in [7] were classified into 7 such as network, SMS, CPU, power, process, memory and virtual memory, and it used a total of 32 features. In particular, the features about memory were classified into memory and virtual memory in detail, and the memory was classified into native, dalvik and other to organize a memory-centered feature set. In addition, it also carried out a performance evaluation for the machine learning classifiers such as RandomForest, Naive Bayes, Logistic and SVM through experiments. However, the Android environment used in the [7]'s experiments was 2.3.3 version, and it needs to acquire the root authority for extracting detailed features of memory for each process such as native, dalvik and other in the current Android environment with a version of above 4.0.

Therefore, this paper selects new Linux kernel based features to detect malware in the Android environment with a version of above 4.0. Most of the newly selected features are classified into memory, CPU and network, and it uses a total of 59 features. In addition, it carries out feature selection to find the optimum of 59 features. The detailed description about the features is provided in Section 3.

III. LINUX KERNEL-BASED FEATURES

This section introduces features to detect malware in the Android with a version of above 4.0. It also introduces feature selection to improve the detection system's performance by removing unnecessary features.

3.1 Linux Kernel-based Feature Extraction

The number of features proposed in this paper is a total of 59, which are extracted on the basis of Linux kernel in the Android. Most of the features are divided into three kinds such as memory, CPU and network, and their details are as Table 1.

Table 1. Linux kernel-based proposed feature

Category	Feature
Memory (24)	MemFree, Buffers, Cached, Active, Inactive, Active(anon), Inactive(anon), Active(file), Inactive(file), Unevictable, HighFree, LowFree, Dirty, AnonPages, Mapped, Shmem, Slab, SReclaimable, SUnreclaim, KernelStack, PageTables, Committed_AS, VmallocUsed, VmallocChuck
CPU (10)	User, Nice, System, Idle, IOW, IRQ, SIRQ, CPU_usage, User_CPU_usage, System_CPU_usage
Network (25)	InReceives, OutRequests, InMsgs, InErrors, InDestUnreachs, OutMsgs, OutErrors, OutDestUnreachs, OutEchos, OutEchoReps, ActiveOpens, EstaResets, CurrEstab, InSegs, OutSegs, RetransSegs,OutRsts, InDatagrams, OutDatagrams, Total_Rxbytes, Total_Txbytes, Real_Rxbytes, Real_Txbytes, Rxbyte_Percentage_change, Txbyte_Percentage_change
The number of total features = 59	

The features introduced above extract contents existed on the '/proc' folder in the Linux kernel. In the '/proc' folder, the mobile device's process information and the device's state information etc. are stored as a file form, and this information is changed in real time to be stored.

Taking a closer look at respective classifications, the features related to memory are obtained by pashing contents existed in the '/proc/meminfo' file. The features related to CPU are obtained by pashing the result of Linux's 'top' command. This paper not only extracts CPU-related features from the result of 'top' command but also calculates the extracted features to represent the total, user's and system's CPU usage as a percentage to also use the result as features.

For the features related to networks, they are extracted by two methods. The features related to Rxbytes and Txbytes use 'tcp_rcv' and 'tcp_snd' files in the '/proc/uid_stat/uid' folder. This extracted Rxbyte(rcv) and Txbyte(snd) information is exploited to additionally calculate the number of Rxbyte and Txbyte generated per unit time and the variation rates of Rxbyte and Txbyte to use them as features. Other network features are obtained by pashing the '/proc/net/snmp' file's content.

3.2 Feature Selection for Optimal Features

This paper proposes 59 features, which is relatively larger in number than the existing studies. If a large number of features are used like this, there could be much overhead in feature extraction of an agent, and analysis time and detection performance of a detection system when extracting features or performing machine learning, so it needs to remove unnecessary features. Therefore, this paper performs feature selection to remove unnecessary features. Fig. 2 shows the feature selection process.

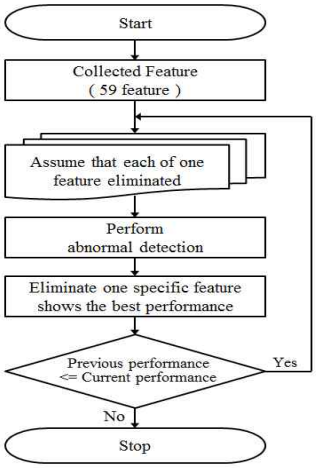

Fig. 2. Workflow in feature selection process

First, it calculates the malware detection result when using the whole features. Then, it assumes a case of removing a certain feature and calculates the malware detection result. In other words, the malware detection experiment is carried out 58 times except the first feature. From the result above, it selects an experiment with the best performance and removes the corresponding feature. Then, it assumes a case of removing another feature other than the removed one to calculate the malware detection result. This feature selection procedure is repeated until the detection performance becomes lower than the previous result. Information about the selected features other than the ones removed after the feature selection is sent to the detection system. The detection system detects the presence of malware through a machine learning classifier.

IV. Experiment and Result

4.1 Experimental Data Set

For the data set used in this paper, the training and test data for the machine learning is created after mixing the feature information under normal situations with the feature information under situations when abnormal applications are executed. Looking at in detail, for the training data, the respective number of vectors under normal situations and under situations when abnormal applications are executed is 2,000 with a one-to-one ratio, and for the test data, the number of vectors under normal situations is 1,000 and the number of vectors under abnormal situations is 250 for each abnormal application. For the test data, the reason that vectors under normal situations is larger in number than ones under situations when abnormal applications are executed is because there are more normal situations than abnormal ones in the actual mobile environment.

For abnormal applications used in the experiment, abnormal applications frequently introduced in the Ahnlab ASEC report [11] were selected, and they are composed of total 6 such as SMSHider, GoldDream, Snake, FakeInst, PjApps and DrawSlasher.

4.2 Experiment Method

The schematic diagram of malware detection system used in this paper is as Fig. 3.

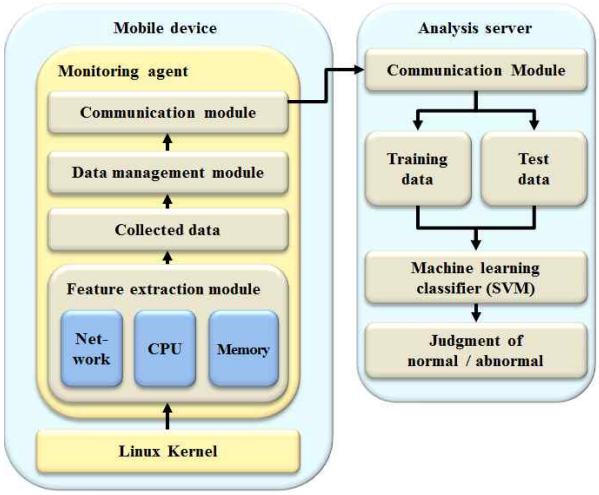

Fig. 3. Structure of abnormal detection system

The analysis system is composed of a monitoring agent and analysis server, and a feature extraction module in the monitoring agent extracts features selected through the feature selection earlier mentioned and sends the collected data to the data management module. Features are periodically extracted every 10 seconds. The data management module vectorizes features collected as a vector, and the vectorized data is sent to the analysis server through the communication module. The analysis server classifies data received from the monitoring agent into the training and test data to store them. The ratio of the training data to test one is 3:2, and the content in the training and test data is not overlapped. This created training and test data is used to detect the presence of malware through the SVM (support vector machine) which is one of machine learning classifiers.

4.3 Experiment Result

4.3.1. Results of Feature selection

This section describes a result of performing the feature selection earlier introduced. To present performance of the feature selection result, this paper uses TPR (true positive rate), FPR (false positive rate), Precision and Accuracy, which are performance indexes used generally in the machine learning. Table 3 shows a result of performing the feature selection.

Table 3. Experiment result of feature selection

The number of eliminated feature	TPR	FPR	Precision	Accuracy	Eliminated Feature
0	95.93%	0.68%	96.26%	98.84%	X
1	95.96%	0.67%	96.39%	98.84%	InErrors
2	95.96%	0.67%	96.39%	98.84%	VmallocUsed
3	95.97%	0.67%	96.41%	98.85%	CurrEstab
			:		
22	95.97%	0.67%	96.63%	98.85%	OutRequests
23	**95.97%**	**0.67%**	**96.63%**	**98.85%**	**Userpercent**
24	95.97%	0.67%	96.46%	98.85%	IOW
25	95.97%	0.67%	96.40%	98.85%	OutDestUnreachs

As shown in Table 3, it could be found that the performance is gradually improved whenever removing unnecessary features one by one, and as a result, it showed the best performance when removing 23 features such as InErrors,

VmallocUsed, CurrEstab, etc. Table 4 shows 23 features removed and 36 ones remained after performing the feature selection. In addition, this paper compared situations before and after performing the feature selection. Before performing the feature selection, the time required for training is 2,590ms, and the time required for testing is 31ms. On the hand, after performing the feature selection, the time required for training is 1,960ms, and the time required for testing is 32ms. It is considered that the difference would become larger when using more malware information in the future.

Table 4. Feature information after feature selection

	The eliminated feature by feature selection (23 feature)	The selected feature after feature selection (36 feature)
Feature information	Active, Active(anon), Active(file), Anonpages, Buffers, Cached, CPUPercent, CurrEstab, Dirty, HighFree, Inactive, Inactive(anon), Inactive(file), InErrors, LowFree, MemFree, Nice, OutRequests, System, SystemPercent, Txbytes_percentage_change, UserPercent, VmallocUsed	ActiveOpens, Commited_AS, EstaResets, Idle, InDatagrams, InDestUnreachs, InMsgs, InReceives, InSegs, IOW, IRQ, KernelStack, Mapped, OutDatagrams, OutDestUnreachs, OutEchoReps, OutEchos, OutErrors, OutMsgs, OutRsts, OutSegs, PageTables, Real_Rxbytes, Real_Txbytes, RetransSegs, Rxbytes_percentage_change, Shmem, SIRQ, Slab, SReclaimable, SUnreclaim, Total_Rxbytes, Total_Txbytes, Unevictable, User, VmallocChuck,

4.3.2. Result of Malware Detection

This section verifies performance and validity of the features extracted earlier through experiments. The experiment compares results of detecting malware when not performing the feature selection and when performing the feature selection. Table 5 shows the result of detecting malware when not performing the feature selection, and when performing the feature selection.

Table 5. Comparison of abnormal detection result

	Before feature selection				After feature selection			
	TPR	FPR	Precision	Accuracy	TPR	FPR	Precision	Accuracy
Normal	71.80%	0.02%	99.86%	95.96%	71.80%	0.00%	100.00%	95.97%
SMSHider	100.00%	1.97%	89.45%	98.31%	100.00%	0.45%	97.37%	99.61%
GoldDream	99.80%	0.10%	99.40%	99.89%	100.00%	0.17%	99.02%	99.86%
Snake	100.00%	0.98%	94.43%	99.16%	100.00%	3.93%	80.91%	96.63%
FakeInst	100.00%	0.00%	100.00%	100.00%	100.00%	0.00%	100.00%	100.00%
PjApps	100.00%	0.15%	99.11%	99.87%	100.00%	0.15%	99.11%	99.87%
DrawSlasher	99.90%	1.53%	91.57%	98.67%	100.00%	0.00%	100.00%	100.00%
Average	**95.93%**	**0.68%**	**96.26%**	**98.84%**	**95.97%**	**0.67%**	**96.63%**	**98.85%**

According to the result of Table 5, before feature selection, the malware detection system's performance is relatively good even when using the whole features. However, as described earlier, it needs to perform the feature selection for removing unnecessary features. On the other hands, after feature selection, the performance is more improved than when using the whole features. Because the malware detection result was relatively high when using the whole features, it did not show significant performance improvement even after performing the feature selection. However, it could be said that the feature selection is necessary in terms of reducing the detection system's overhead by removing unnecessary features.

V. CONCLUSION

This paper proposed new Linux-kernel based features to detect malware in the environment of Android with a version 4.0 or higher. For the newly proposed features, various features such as variations were additionally suggested by not only monitoring numerical values of the corresponding features but also calculating the extracted features. Furthermore, the existing studies considered only Rxbyte and Txbyte for the network-related features, but this paper additionally considered features related to IP, TCP, ICMP and UDP protocols as well as Rxbyte and Txbyte.

In addition, it performed the feature selection to reduce the detection system's overhead by removing unnecessary features from the proposed features. As a result of the feature selection, 23 features of 59 ones were removed so that 36 ones were selected. To verify validity of the feature selection result, the SVM, which is one of machine learning classifiers, was used to carry out experiments. The experiment result showed a little bit improved, and it could be considered to show relatively better results. In addition, comparison was carried out in terms of resource consumption before and after performing the feature selection.

In the future, it would like to introduce a variety of malicious applications other than the malicious applications used in the experiment to prove validity of the malicious application detection results, and plans to perform a malware detection study through time-series analysis methods other than machine learning based methods.

ACKNOWLEDGEMENT

This research was supported by Next-Generation Information Computing Development Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Science, ICT & Future Plannig (2010-0020723).

REFERENCES

[1] A. Shabtai, L. Tenenboim-Chekina, D. Mimran, L. Rokach, B. Shapira, Y. Elovici, "Mobile malware detection through analysis of deviations in application network behavior.", Computers & Security, 2014.

[2] Hyo-Sik Ham, Hwan-Hee Kim, Myung-Sup Kim, Mi-Jung Choi, "Linear SVM-based android malware detection", Frontier and Innovation in Future Computing and Communications. Springer Netherlands, 2014.

[3] F-Secure, "Mobile Threat Report 2013", 2013.

[4] Uddin, Mueen, Kamran Khowaja, et al. "Dynamic Multi-Layer Signature Based Intrusion Detection System Using Mobile Agents", International Journal of Network Security & Its Applications, 2010.

[5] Rastogi, Vaibhav, Yan Chen, William Enck., "Appsplayground: Automatic security analysis of smartphone applications.", Proceedings of the third ACM conference on Data and application security and privacy, 2013.

[6] Shabtai, Asaf, et al. "Andromaly: a behavioral malware detection framework for android devices" Journal of Intelligent Information Systems, 2012.

[7] Hyo-Sik Ham, Mi-Jung Choi, "Analysis of Android Malware Detection Performance using Machine Learning Classifiers", International Conference on ICT Convergence, October, 2013.

[8] Rastogi, Vaibhav, Yan Chen, William Enck. "Appsplayground: Automatic security analysis of smartphone applications." Proceedings of the third ACM conference on Data and application security and privacy, 2013.

[9] Dai-Fei Guo, Ai-Fen Sui, Yi-Jie Shi, Jian-Jun Hu, Guan-Zhou Lin and Tao Guo, "Behavior Classification based Self-learning Mobile malware detection", Journal of Computers, 2014.

[10] Brandon Amos, Hamilton Turner, Jules White, "Applying Machine Learning Classifiers to Dynamic Android Malware Detection at Scale", Wireless Communications and Mobile Computing Conference, 2013 9th International. IEEE, 2013.

[11] Ahnlab, "Ahnlab ASEC Report", 2013.