Detection of Malicious Android Mobile Applications Based on Aggregated System Call Events

You Joung Ham and Hyung-Woo Lee

***Abstract*—The diverse types of mobile applications are used regardless of time and place, as a number of Android mobile device users have been recently increased. However, the breach of privacy through illegal leakage of personal information and financial information inside mobile devices has occurred without users' notices, as the malicious mobile application is relatively increasing In order to reduce the damage caused by the malicious Android applications, the efficient detection mechanism should be developed to determine normal and malicious apps correctly. In this paper, we aggregated real-time system call events activated from malware samples distributed by Android *Malware Genome* Project. After extracting the basic difference feature and characteristics of system call events pattern from each normal and malicious applications, we can determine whether any given anonymous mobile application is malicious or normal one.**

***Index Terms*—Android, malicious mobile applications, system call events, similarity.**

I. Introduction

The procedural analysis reveals that the user devices will get infected with malicious codes and lead to the problems rerouting key information to external servers with which intruder specified through changes of access permission, once users run the programs which were downloaded from open market or black markets [1], [2]. Mobile malicious apps based on Android which leaks the personal and financial information by causing malfunction and consuming the batteries of devices have consistently been increasing [3]. Therefore, techniques [4], [5] monitoring malicious app events have been presented to detect the intrusion toward mobile devices in a bid to reduce damages through spread of malicious app like this, but mechanism should be developed to discriminate malicious apps from normal apps of commercial mobile devices.

Detection methods for attacks on mobile devices [6], [7] have been proposed to reduce the vulnerability from malicious mobile apps. However, an advanced mechanism that provides more enhanced ways of classifying malicious apps on common mobile devices should be developed [8], [9]. In first, it is necessary to analyze the attack mechanism based on the recent security vulnerabilities of Android-based mobile devices, and analyses the characteristics of malicious apps with activation pattern using Linux based *Strace* tool [10] on Android Platform. Therefore, we want to suggest a method to distinguish Android-based malicious apps based on the system call event pattern internally activated after running suspicious malicious applications.

We analyzed the malicious system call event pattern selected from Android *Malware Genome* Project [11]. The actual system call patterns are extracted from the normal and malicious apps on Android-based mobile devices. And then, feature events were aggregated to calculate a similarity analysis between normal and malicious event set. Based on it, we can extract characteristics of system call event pattern of malicious apps. Based on these characteristics, we can determine whether any given anonymous mobile application is malicious or normal one.

Manuscript received December 15, 2013; revised February 14, 2014. This work was supported in part by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education, Science and Technology (Grant # 2012R1A1A2004573).

You Joung Ham and Hyung Woo Lee are with Dept. of Computer Engineering, Hanshin University, Rep. of Korea (e-mail: you86400@hanmail.net, hwlee@hs.ac.kr).

II. Malicious Android Mobile Applications and Existing Mechanism

A. Implicit Malicious Code

Malicious code infected apps have been spread through open or non-public market by group of anonymous developers. The reason that the security threats targeting the Android platform are increasing is based on the fact that the Android platform provides functions that are easily accessed by allowing various forms of attacks to occur based on its openness and portable features [12]. Therefore, the security vulnerabilities of the Android platform are causing various types of attack. As in Fig. 1, attackers hide a malware by activating exploits from inside a mobile app that appears normal in order to distribute it to common user's smartphone. Users execute an installed app that includes the malware silently leaking personal information stored inside of device to get root permission and privilege. When the malware is executed, the attacker gets illegal access to the internal resource without user's notification.

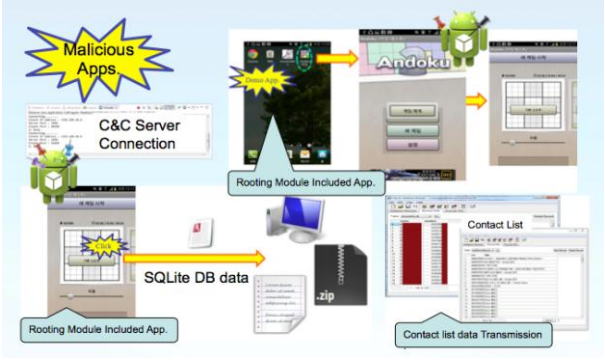

Fig. 1. Attack executed by implicit malicious code.

DOI: 10.7763/IJCCE.2014.V3.310

B. Existing Crowdroid System

Existing *Crowdroid* [6] system used *Strace* tool which can be run on Android based kernel in a bid to collect system call events from mobile devices. *Strace* creates the output file using the collected system call events that are traced at being invoked when Android applications are run. The monitoring results of system call events to Android kernel using *Strace* are transferred to remote activity based malicious software detection servers. The structure of *Crowdroid*'s activity based malicious software detection framework is drawn at Fig. 2.

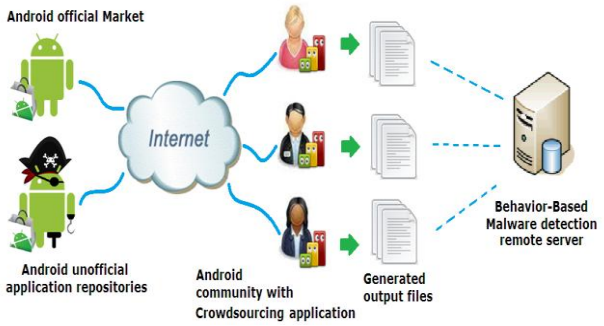

Fig. 2. Crowdroid : existing activity driven malicious software detection framework.

The system call event information created by running mobile apps after Crowdsourcing apps are installed at normal users' mobile devices is transferred to activity based malicious software detection servers. The detection process of malicious software based on information provided by Android community users who installed *Crowdroid* app based on the framework in Fig. 2 has been shown at Fig. 3.

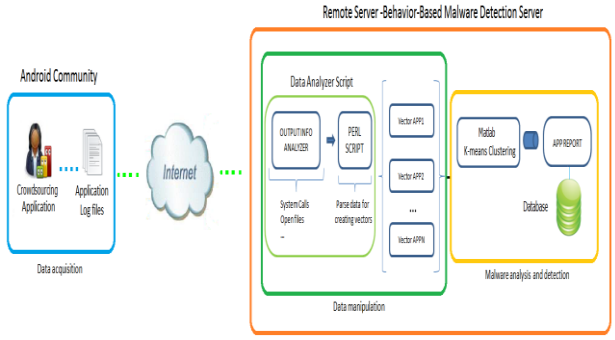

Fig. 3. Detection process of Android malicious software.

However, problems came from discrimination of malicious codes conducted only with similarity analysis on system call events occurring on executing normal and malicious apps without informing of which system call events are invoked. Therefore, we have extracted patterns on system call events occurring from apps containing malicious codes, and designed and implemented techniques to discriminate the malicious codes depending on them by conducting similarity analysis on system call events occurring on multiple mobile devices which installed Android platform.

III. Aggregation of System Call Events

We aggregated and analyzed the system call events automatically from Android based mobile devices with customized *Strace* module. It collects system call events internally occurring on executing normal and malicious apps from multiple commercial mobile devices based on Android platform, and it made the malicious applications detectable based on this as Fig. 4. It also analyzed similarity with malicious apps by analyzing occurrence, call correlation and sequence of system call events from normal apps running on commercial mobile devices in real-time. It transfers system call events to servers after checking out the events which occur on executing apps through adb shell.

Fig. 4. Activation pattern based mobile application determination method based on system call events.

As shown in Fig. 5, we developed system call event-monitoring procedure that enables the user to automatically retrieve activated services and processes running on inside of Android kernel. If Activation Monitoring application is executed, it invokes system call event-monitoring daemon at kernel and the application is running in background as Fig. 5. The background running application transfers event information to DB server in a log format whenever event occurs. DB server is implemented to collect and store the event data that comes from multiple devices. Those events information collected from the mobile device are used to detect any suspected event due to malicious exploits.

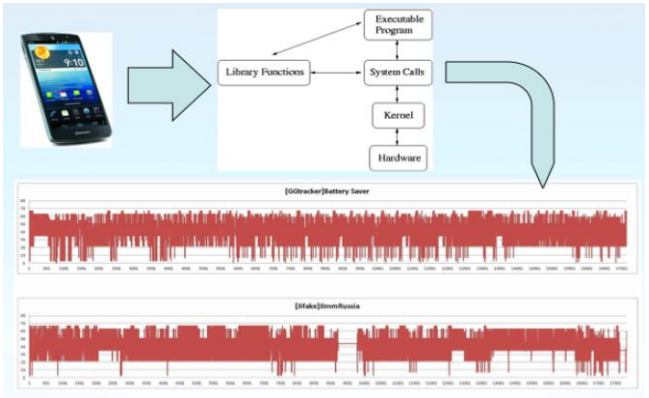

Fig. 5. Activation pattern aggregation and monitoring.

We could measure correlation between normal and malicious apps through checking what kind of system call event functions exist in two groups of apps and how frequently they show up. We also could extract system call events in Android kernel by using *Strace* while normal and malicious applications are running. Based on these data sets, we compared the patterns of internal event activated from Android mobile device. (See Fig. 6).

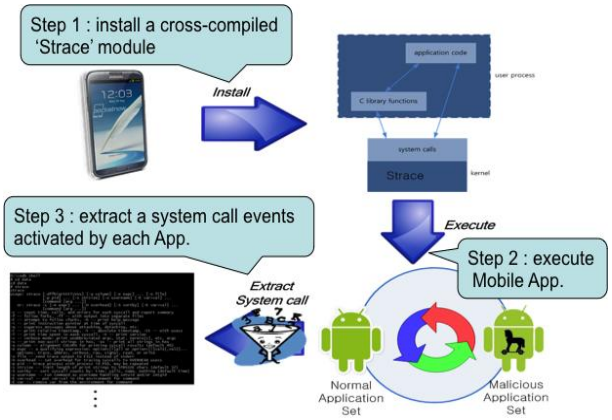

Fig. 6. Activation pattern aggregation steps.

Using proposed mechanism, we can aggregate system call events as follow figure. After install cross-compiled *Strace* module on mobile device, we can execute any application to extract system call events activated by each application as Fig. 7.

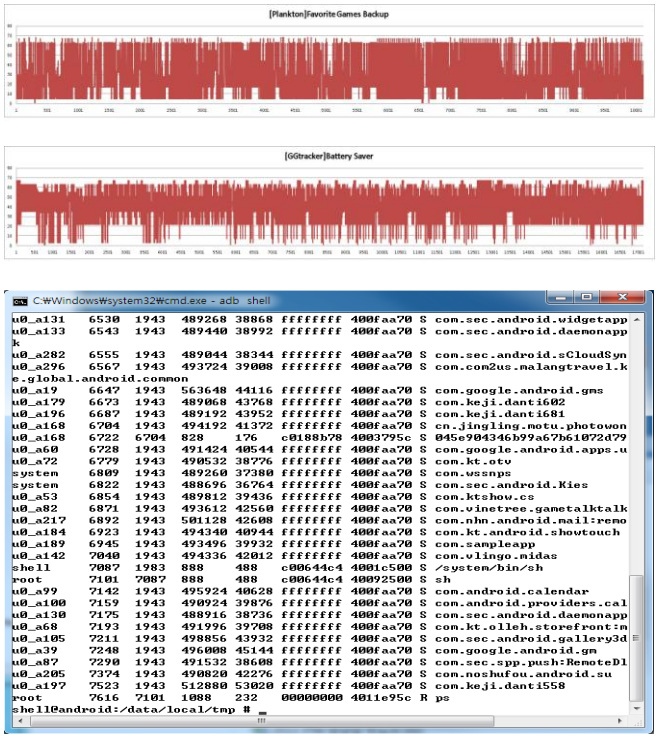

Fig. 7. Activated system call events.

IV. System Call Events from Malicious Applications

A. Characteristics of Malicious Android Applications

Android *Malware Genome* Project categorizes 1,260 samples of malicious apps largely by their characteristics to Malware Installation, Activation, Malicious Payloads and Permission Uses [9]. Additionally, we can classify malicious app into Repacking, Update-Attack, Drive-by Download and Standalone [11]-[13] based on its internal activity pattern.

Repacking is a method that repacks an application, in which the malicious developer downloads an application that has been registered by online such as in Android official market, inserts a malware, which has been modified from apk or jar file, and distributes it. Disguising as a normal application, it leaks personal or financial information of users by causing damages often [9], [14], [15]. And Update attack is a method that installs a malicious app when a user downloads an update. It cannot only install an app that the user don't know but also leak private information or lead to billing. Also, update attack has a self-update feature, and can be divided into four main techniques [9]. Drive-by download is a form of remote attack that downloads and executes a malware without the user knowing and mainly user-after-free and Heap Spraying method attack cases are found. Drive-by-download has a long patch cycle, so the relevant vulnerability is attacked before the patching, allowing leakage of user's private information. This Drive-By-Download attack also, like the update attack, is difficult to detect compared to other malicious apps. Standalone is a type of app that runs by itself without help from any different tool. These applications use a route attack that makes a detour around the internal security sandbox without asking the user [14], [16]-[19].

B. Characteristics of Malicious System Call Events

Among the malicious app, BaseBridge and ArtifactDataCable app can change the Wi-Fi option without the user knowing and another application is additionally installed to damage by leaking SMS, personal information, call records or causing billings when the app starts [20]-[23]. We analyzed malicious application events targeting the apk file contained in Update attack malware. In case of Update Attack, we can find a pattern that system call events of fchown32, fdatasync, mkdir, rmdir, statfs64 and umask which were relatively hard to be found in the earlier system call events of normal application. These system call events can be used to create or delete directory or have features of synchronization of data in file disk, obtaining file system information, creation of file mask, therefore these events were used to causing damage on user's device.

System call events of bind, brk, connect, msgget, recv, recvfrom, select, semget, semop and setsockopt were relatively hard to be found in the earlier system call events of Update Attack. The occurred system call events might be used to change the size of data segments in the process, or return the identification number of message queue and read the data and message from socket. 17 system call events of bind, brk, connect, fchown32, fdatasync, fsync, mkdir, msgget, recv, recvfrom, rmdir, select, semget, semop, setsockopt, statfs64 and umask, which do not occur in normal application, are found in malicious applications. Therefore, any given apps could be suspected as malicious mobile application if the 17 kinds of system call events above have occurred simultaneously in the application.

Based on this, we can find a pattern that the system call events such as bind, brk, connect, fdatasync, mkdir, msgget, pwrite, recv, recvfrom, rename, rt_sigreturn, semget, semop, setsockopt, socket, statfs64, SYS_224 and SYS_248 were occurred only in malicious application. And 11 system call such as chdir, flock, getcwd, nanosleep, poll, prctl, rt_sigreturn, sigaction, SYS_281 and SYS_283 were occurred only in normal application. And those occurred both in normal and malicious app were 40 including access, chmod and clock_gettime as follow Fig. 8.

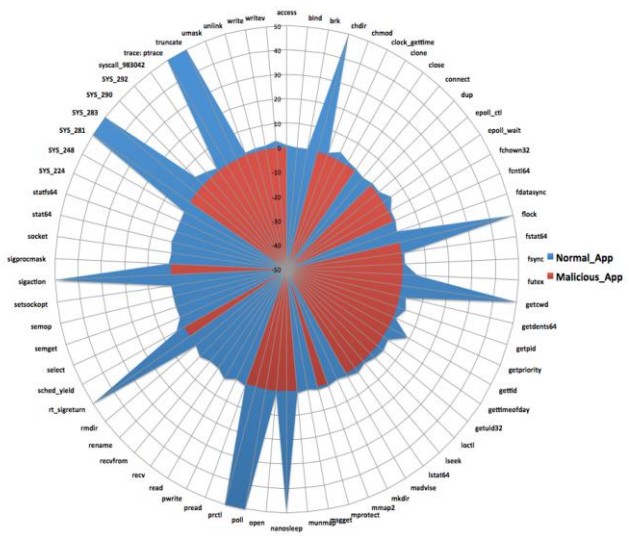

Fig. 8. Activity pattern comparison between normal and malicious application.

V. Determination of Malicious Applications

A. Distinction Procedure Based on Activity Pattern

We can construct a determination method between normal and malicious application events based on the event analysis on normal and malicious application described previously. When a user downloaded and installed a found app, it is difficult to distinguish whether the app is malicious or not. This paper proposed a distinction method for installed apps based on system call events. In first, APK file installed in the mobile device should be analyzed in order to distinguish whether the installed app is malicious or normal by decompressing and obtaining the assembly code. Extracting application permission from AndroidManifest.xml in the decompressed folder and analyzing them based on normal permission group and malicious permission group can distinguish whether the given app is normal or malicious. Algorithm that distinguished malicious app through the similarity of activity pattern aggregated from apps can be illustrated as Fig. 9.

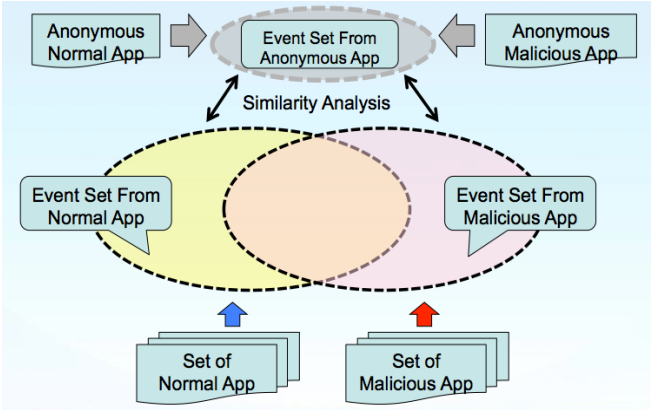

Fig. 9. System call event based similarity analysis procedure.

Installing a found app, obtaining permissions contained in the apps and extracting events using *Strace* can draw out similarity with normal event group and malicious event group. We randomly chose 10 apps categorized normal and malicious app. Through which, *SimAppNml* (Similarity to Normal) and *SimAppMal* (Similarity to Malicious) were drawn through this and the similarity equation, *(k-SimAppNml)*(SimAppMal *k (i*=1, 2,…,*n)* provides the final result of distinguishing normal or malicious. Here, *S(Similarity)* ranges from 0 to 1, and k represents the weighted value. We can see the difference pattern on activated system call event aggregated each from normal and malicious applications as follow Fig. 10. Based on this pattern, we can determine whether any application is malicious or not.

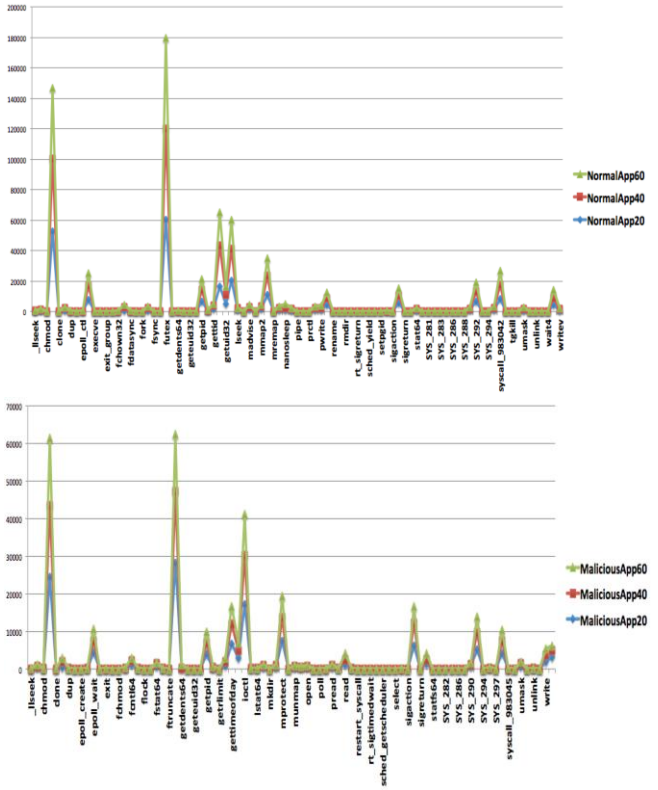

Fig. 10. Pattern of normal and malicious system call event.

B. Malicious Application Distinction

Frequent normal event group consists of 32 events that occurred only in the previously addressed normal apps including clock_gettime, epoll_wait, getcwd and poll, and those that occurred more often in normal apps among those that occurred in both normal and malicious apps. On the other hand, malicious event group also consists of 36 events including bind, pwrite, rename and unlink, and those that occur relatively more often in malicious apps among those that occurred in both normal and malicious apps. As the result, the normal and malicious similarity of normal apps and malicious apps based on normal event groups and malicious event groups can be illustrated as a graph in Fig. 11 and Fig. 12 respectively as below.

SimAppNml, the number of normal app events that correspond to normal event group, was confirmed to give relatively higher value in normal apps than in malicious apps. The reason is because the normal event group is a group of events that occur both in normal and malicious apps but occur more often in normal apps. On the other hand, *SimAppMal*, the number of malicious and normal events that correspond to the malicious event group over the number of malicious app events, gave much higher value in the malicious apps that in

normal apps because the event group occurred only in malicious apps will be have a malware.

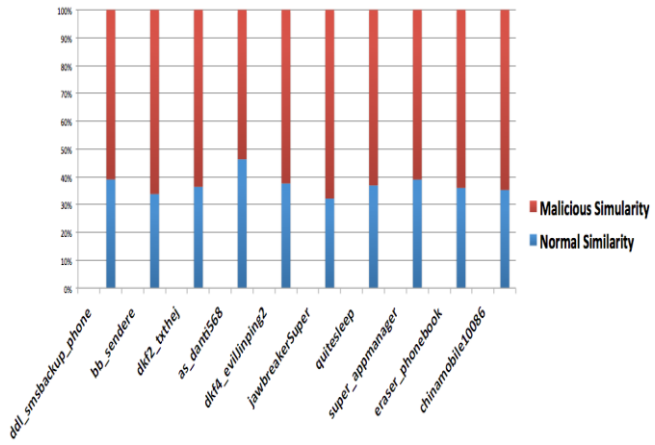

Fig. 11. Result of normal and malicious similarity.

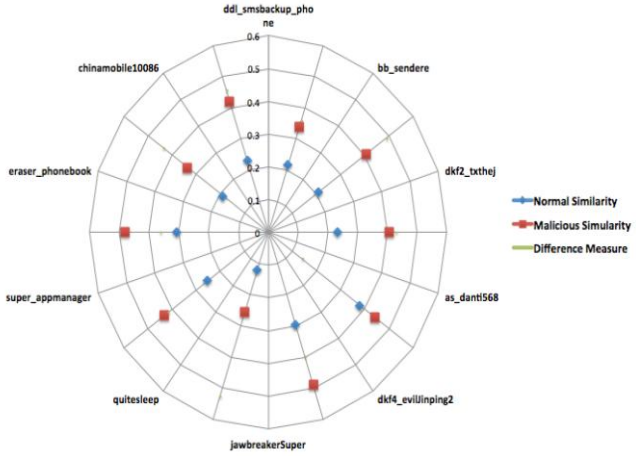

Fig. 12. Normal and malicious similarity calculation to distinct malicious attack.

In order to make a more precise distinction on whether the events found in some apps are closer to malicious, we can use similarity equation. We have suggested a malicious app distinction method by assign weighted value k to activated event in the malicious event group to distinguish malicious app through the analyzed similarity. Malicious events that correspond more to the malicious event group occurred more than normal app events. Therefore, *SimAppMal*, to which the weighted value of k is multiplied, gives much larger value than *SimAppNml* does. As above, malicious app distinction similarity showed a high value for a given malicious application. Therefore this equation could be expected to show the same result when applied to a found app in another mobile environment.

C. Comparison with Existing Mechanism

The similarity and differences between the suggested technique at chapter two of this paper and *Crowdroid* technique [6] are shown in follow Table I. Both mechanism collected and analyzed the system calls invoked from Android platforms on the basis of Linux kernel via using *Strace* module. However, we conducted the analysis on frequencies and similarity of system call functions occurring on executing Strace for the commercial mobile devices based on Android platform.

TABLE I: PERFORMANCE COMPARISON

Comparison	*Crowdroid* System	Proposed Mechanism
Experimental Platform	Android	Android
System Call Event Extraction Method	Strace	Strace
System Call Event Logging	○	○
System Call Event Aggregation	1 device	n device
Real-Time Event Aggregation	×	○
Apps Event Feature Analysis	△	○
Malicious Apps Discrimination Algorithm	K-means	Frequency & Euclidean Distance
Malicious Apps Decision Function	○	○
Malicious Apps Event Sequence Extraction	×	○

VI. CONCLUSIONS

This study presented techniques to effectively detect the malicious apps which are easy to install and use on its Android based commercial mobile device environment. Above all, it analyzed the access methods and research results on *Crowdroid* techniques collecting and analyzing the system call events occurring upon executing apps. It suggested techniques of discriminating the malicious apps based on this, implementing the extracting module of the system call events in Android based commercial mobile devices. It performed comparison analysis on characteristics of system call events occurring on normal and malicious apps using *Strace* module being able to collect the system call events in Android kernel. It also presented the algorithm to discriminate the malicious apps using the algorithm of frequency and similarity analysis of occurring events. The use of techniques presented in this study made it possible to analyze the characteristics of system call events occurring upon executing malicious apps, and can be applied for a way to discriminate whether the arbitrary mobile apps are malicious or not through this.

This paper targeted game applications of Google Play Store as normal and 1,260 malicious samples distributed by Android *Malware Genome* Project as abnormal, and proposed an effective method for distinguishing malicious apps in Android-based common mobile device environment. Use of the method proposed in this paper could analyze the characteristics of system call events that occurred when normal apps and malicious apps were in action, which could be applied to a method of distinguishing whether any given app is malicious. Also, the sequence analysis based on system call events extracted from *Strace* could draw out a system function that occurs both in normal and malicious apps with more frequent occurrence in malicious apps and relatively less frequent occurrence in normal apps.

ACKNOWLEDGMENT

This research was partially funded by the MSIP (Ministry of Science, ICT & Future Planning), Korea in the ICT R&D Program 2013.

REFERENCES

[1] Google play android market. [Online]. Available: https://play.google.com/store/apps

[2] More than 700,000 malicious Android apps wreak havoc on the web. [Online]. Available: http://www.neowin.net/news/more-than-700000-malicious-apps-wreak-havoc-in-the-play-store

[3] Android (operating system). [Online]. Available: http://en.wikipedia.org/wiki/Android_(operating_system)

[4] Uncovering android master key that makes 99% of devices vulnerable. [Online]. Available: http://bluebox.com/corporate-blog/bluebox-uncovers-android-master-key/

[5] W. Enck, D. Octeau, P. McDaniel, and S. Chaudhuri, "A study of android application security," in *Proc. the 20th USENIX Security Symposium*, 2011.

[6] I. Burquera, U. Zurutuza, and S. Nadjm-Tehrani, "Crowdroid: behavior-based malware detection system for Android," in *Proc. the 1st ACM workshop on Security and privacy in smartphones and mobile devices*, 2011, pp.15-26.

[7] T. E. Wei, C. H. Mao, A. B. Jeng, H. M. Lee, H. T. Wang, and D. J. Wu, "Android malware detection via a latent network behavior analysis," presented at IEEE 11th International Conference on Trust, Security and Privacy in Computing and Communications, 2012.

[8] D. J. Wu, C. H. Mao, T. E. Wei, H. M. Lee, and K. P. Wu, "Droidmat: android malware detection through manifest and api calls tracing," presented at 2013 7th Asia Joint Conference on Information Security, 2013.

[9] Y. J. Ham, W. B. Choi, H. W. Lee, J. D. Lim, and J. N. Kim, "Vulnerability monitoring mechanism in Android based smartphone with correlation analysis on event-driven activities," in *Proc. 2012 2nd International Conference on Computer Science and Network Technology*, 2012, pp. 371-375.

[10] Strace. [Online]. Available: http://en.wikipedia.org/wiki/Strace

[11] Y. J. Zhou and X. X. Jiang, "Dissecting Android malware: characterization and evolution," in *Proc. the 33rd IEEE Symposium on Security and Privacy*, 2012, pp. 95-109.

[12] X. X. Jiang and Y. J. Zhou, *Android Malware*, NY, USA: Springer, 2013.

[13] M. Nauman, S. Khan, and X. Zhang, "Apex: extending android permission model and enforcement with user- defined runtime constraints," in *Proc. the 5th ACM Symposium on Information, Computer and Communications Security*, 2010, pp. 328-332.

[14] A. PorterFelt, M. Finifter, E. Chin, S. Hanna, and D. Wagner, "A survey of mobile malware in the wild," in *Proc. the 1st Workshop on Security and Privacy in Smartphones and Mobile Devices*, 2011, pp. 3-14.

[15] W. Zhou, Y. Zhou, X. Jiang, and P. Ning, "Droidmoss: detecting repackaged smartphone applications in third- party android marketplaces," in *Proc. the 2nd ACM Conference on Data and Application Security and Privacy*, 2012, pp. 317-326.

[16] D. James, *Android Game Programming For Dummies*, Hoboken, New Jersey: John Wiley & Sons, Inc., 2013.

[17] Z. Mednieks, L. Dornin, G. B. Meike, and M. Nakamura, *Programming Android, O'Reilly*, CA: Sebastopol, 2012.

[18] A. Shabtai, Y. Fledel, U. Kanonov, Y. Elovici, and S. Dolev, "Google android: a state-of-the-art review of security mechanisms," Technical Report, Cornell University, 2009

[19] M. Spreitzenbarth and F. Freiling, "Android malware on the rise," University of Erlangen, Dept. of Computer Science, Technical Reports, April 2012.

[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] Y. Zhong, H. Yamaki, and H. Takakura, "A malware classification method based on similarity of function structure," in *Proc. 12th International Symposium of Applications and the Internet*, 2012, pp. 256-261.

[22] S. Bugiel, L. Davi, A. Dmitrienko, T. Fischer, and A. R. Sadeghi, "XMandroid: a new android evolution to mitigate privilege escalation attacks," *System Security Lab, Technische University Darmstadt, Technical Report*, June 2011.

[23] Y. J. Ham, D. Moon, H. W. Lee, J. D. Lim, and J. N. Kim, "Activation pattern analysis on malicious android mobile applications," in *Proc. 2013 Firnst International Conference on Artificial Intelligence, Modelling & Simulation*, 2013, vol. 1, no. 1, pp. 98-103.

You-Joung Ham received B.S degree in information communication from Hanshin University, Gyeong-gi Province, Rep. of Korea in 2012. And she is currently a M.S candidate in Dept. of Computer Engineering from Hanshin University, Gyeong-gi, Korea. Her research interests include secure android application, smart fuzzing and network security.

Hyung-Woo Lee received B.S, M.S and Ph.D. degrees in computer science from Korea University, Seoul, Rep. of Korea, in 1994, 1996, and 1999 respectively. Currently he is a professor at the Department of Computer Engineering, Hanshin University, Gyeong-gi Province, Rep. of Korea. He is a member of KIISC, KSII on Korea Society. Additionally, he is a member of IEEE. He is a program committee chair of WISA 2007, CISC-S 2010 and a review member of IEEE. His research interest is in the areas of network security, cryptography and computer forensics.