International Journal of Security and Its Applications
Vol.8, No.1 (2014), pp.231-246
http://dx.doi.org/10.14257/ijsia.2014.8.1.22

Android Mobile Application System Call Event Pattern Analysis for Determination of Malicious Attack

You Joung Ham[1], Daeyeol Moon[1], Hyung-Woo Lee[1],
Jae Deok Lim[2] and Jeong Nyeo Kim[2]

[1]*Dept. of Computer Engineering, Hanshin University, 411, Yangsan-dong, Osan, Gyyeonggi, 447-791, Rep. of Korea*
[2]*Cyber Security-Convergence Research Laboratory, ETRI, 218, Gajeongno, Yuseong-gu, Daejeon, 305-700, Rep. of Korea*
hwlee@hs.ac.kr

Abstract

Due to the openness of the Android-based open market, the distribution of malicious applications developed by attackers is increasing rapidly. In order to reduce the damage caused by the malicious applications, the mechanism that allows more accurate way to determine normal apps and malicious apps for common mobile devices should be developed. In this paper, the normal system call event patterns were analyzed from the most highly used game app in the Android open market, and the malicious system call event patterns were also analyzed from the malicious game apps extracted from 1260 malware samples distributed by Android MalGenome Project. Using the Strace tool, system call events are aggregated from normal and malicious apps. And analysis of relevance to each event set was performed. Through this process of analyzing the system call events, we can extract a similarity to determine whether any given app is malicious or not.

Keywords: *Android, System call events, Similarity analysis, Mobile Application, Malicious App., Event pattern*

1. Introduction

Recently various forms of apps are developed to distribute through the Android market as the common Android-based smart device users have been increasing [1]. However, the distribution of malicious apps developed by attackers as well as that of general normal apps is increasing due to the openness of the Android-based open market that allows anyone to develop and distribute [2]. Attackers insert a malware in an app and distribute it to common users through the open market or internet targeting common smart work devices equipped with the Android platform [3]. This can attempt an attack that leaks to external server where personal information is stored in smart work devices of common users, such as SMS and phone numbers, as well as the financial information [4]. As described, the security problem is growing significant in the user environment of the Android-based common smart work devices, which is recently widespread in the nation and worldwide [5].

Detection methods for attacks on mobile devices [6-9] have been proposed to reduce the damage from the distribution of malicious apps. However, a mechanism that provides more accurate ways of determining normal apps and malicious apps on common mobile devices must be developed. This paper first analyses the attack types through the recent security vulnerabilities of Android-based mobile devices, extracts events of normal apps and malicious apps, using Android-based event analysis tool, *Strace* [10], and analyses them to

ISSN: 1738-9976 IJSIA

propose a method that analyses the characteristics of normal apps and malicious apps and the event pattern relevance. This paper aims to suggest a method to distinguish Android-based normal apps and malicious apps based on this proposed method.

To describe specifically, this paper analyzed the normal event pattern of the most highly used game apps in the Android open market to find the event pattern of normal apps and malicious apps of Android platform-based mobile devices. The analysis of normal system call event characteristics first targeted the top 80 normal game apps in the game category because malicious app types disguised as normal are most often found among the apps in the game category of the official Android market, *Google Play* [1]. This paper also analyzed malicious event patterns from malicious apps and disguising malicious apps of game app types among the 1260 malicious samples distributed by *Android MalGenome Project* [11]. As described, the implemented experiments extracted normal app and malicious app events from the normal apps and malicious apps of Android-based mobile devices, using the Linux-based system call extraction tool, Strace. First, events that occur when the normal apps and malicious apps are in action were collected to perform a relevance analysis between each event set. This process of analyzing event occurrence characteristics, pattern and distribution on normal apps and malicious apps drew out the event similarity, through which the process of distinguishing malicious apps was implemented.

The structure of this paper is as follows. The second chapter analyzed the recent existing Android-related security vulnerabilities. The third and fourth chapters analyzed the each event pattern of normal apps and malicious apps. The third chapter categorized normal apps, which are the analysis subjects, based on the categories and permissions and analyzed event patterns that occur when each app is in action and their relevance. The fourth chapter analyzed the operation method of main malicious apps and analyzed event patterns that occur when each app is in action and their relevance. The fifth chapter compared the event pattern characteristics of normal apps and malicious apps based on the analysis results from the third and fourth chapter, and proposed a method that can distinguish whether any given app is malicious based on the event similarity, and the sixth chapter presented the conclusion.

2. Vulnerability of Android Platform

Malware targeting smart work devises based on the Android platform are being widely distributed through the open market [12]. 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 open and portable features. Therefore, the security vulnerabilities of the Android platform are causing various forms of attack. And in order to solve this, user-centric active diagnosis for the vulnerabilities and security mechanisms that prevents leakage of personal information are needed.

As in Figure 1, attackers hide a malware inside a mobile app that appears normal in order to distribute it to smart work devices of common users. Users execute the app that includes the malware by processes such as clicking a button, and leakage of personal information saved in the smart work device can occur in this process. When the malware is executed, the attacker wins illegal access to the resource inside the device without user knowing.

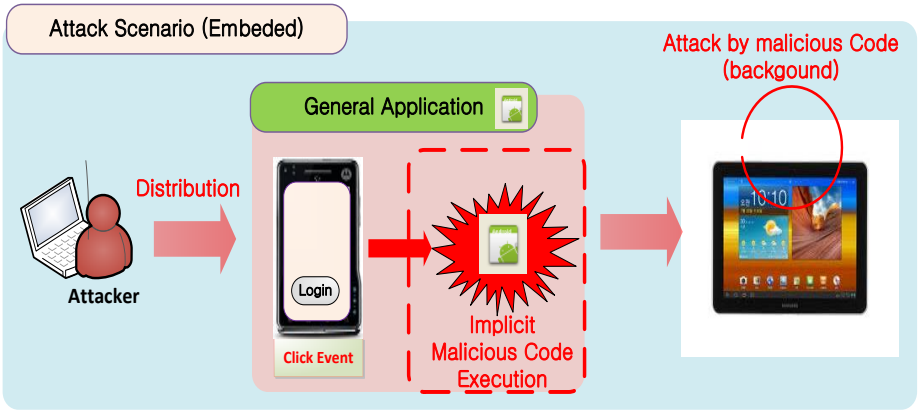

Figure 1. Attack Executed by Implicit Malicious Code

What becomes a more severe problem is when a malware is downloaded and installed from a certain server while the malicious app distributed by an attacker is started, which will not be detected by common basic mobile vaccines.

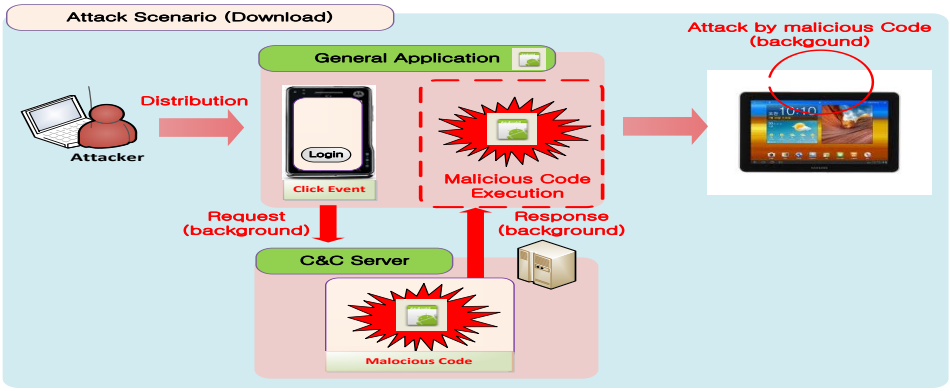

Figure 2. Attack Executed by Installation of Downloaded Malicious Code Inside Normal Application

For example, an attacker can attempt a roundabout connection through SMS and others in order to induce users to download additional malwares from C&C server. Distribution of malware in the Android platform takes various methods other than this, and the inner code of disguised malicious apps is as shown in Figure 3. Malicious app that is disguised as an application for translating SMS received in Android-based smart work devices to a certain language can use malwares related to information collection to execute an attack that leaks the personal information such as SMS records to external C&C server.

Another attack that violates personal information of user is an unauthorized leakage of user's location information. This malicious application executes the attack of leaking user's location information without the user knowing. Other malicious applications cause financial damage by executing online banking attacks while disguising as a normal mobile security application and communicating with C&C server. The attacks on online banking process available with smart work devices can cause damage such as taking user's password while disguising as a normal application.

Figure 3. Malicious App that is Disguised as a Normal Application that Translates Received SMS to Certain Language

Figure 4. Unauthorized Leakage of User's Location Information by Malicious App

To fight against this, mobile vaccine programs for reducing security vulnerabilities to Android-based malicious apps have increased. '*V3 Mobile*' developed by *AhnLab* [13] and *Eastsoft*'s A*Lyac Android* [14] are Android-based mobile vaccine programs that detect and block smishing attacks of malicious apps and provide a feature that detects repackaged apps. Also, *Kaspersky Internet Security for Android* [15] provides blocks on malwares and virus, inspection at all or preset time and filtering on calls and SMS services. *BullGard Smart Mobile Security* [16] provides detection and block of virus and spyware for SMS, MMS, email, bluetooth and so on, and allows detection and remote control through GPS in case of loss or theft.

However, as noted earlier, the need to analyze inner system call events that occur any time in the operation process of each application becomes evident in order to distinguish the normal mobile apps and the malicious available on Android-based smart work devices. Therefore this paper analyzed the event pattern that occurs in the operation of normal and malicious apps to propose a method to determine whether any given app is malicious.

3. Normal Game App System Event Pattern Analysis

Malicious apps disguised as normal apps are found most often among the apps in the game category of official *Android Google Play Store*. Therefore, the analysis on normal system call event first targeted the top 80 normal game apps in the game category. Specifically, seven subcategories in the game category have been analyzed based on permission to classify the characteristics of system call event of normal applications.

3.1. Category based System Call Event Analysis

Normal game apps have been analyzed by categories. System call events that occur when a normal game app is running and their characteristics have been analyzed by targeting seven subcategories within the game category.

3.1.1. Game Applications for Analysis

Game category within *Google Play Store* has seven subcategories, among which the most highly used 80 normal game apps can be categorized as 8 subgroups. Event analysis first targeted mobile game applications that are included in the arcade category such as *Maldalija for Kakao* along with about 25 other applications. Event analysis then targeted applications included in the casual game category such as *Everybody's Marble* along with about 30 other applications, and the brain game and puzzle category that include *Anipang* and about 11 other applications. In addition, applications in the car race, sports game, entertainment and card game categories were targeted for normal even pattern and characteristic analysis.

3.1.2. App Event Analysis by Categories

Events of normal game applications in each category have been analyzed by *Strace* instead of the classic method based on normal game applications in the seven categories described above.

Event call counts of normal game application in the arcade category can be illustrated as Figure 5 below. The result of analysis on event distribution of 25 normal game applications in the arcade category indicated that eight events of *clock_gettime, epoll_wait, futex, getpid, ioctl, mprotect, read* and *SYS_292* occurred relatively more often than other events.

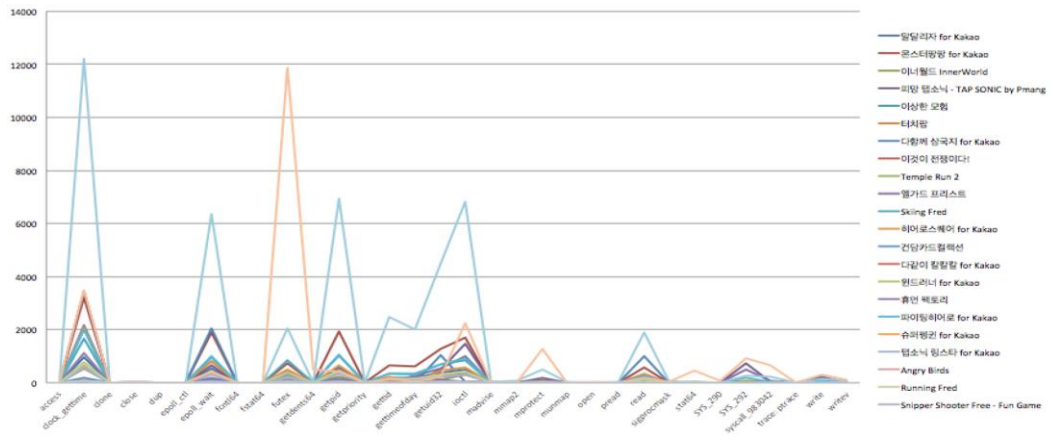

Figure 5. Normal App System Call Pattern: Arcade

Event call counts of normal game application in the casual game category can be illustrated as Figure 6 below. The result analysis on event distribution of 30 normal game applications in the casual game category indicated, similar to the arcade category events, that nine events of *clock_gettime, epoll_wait, futex, getpid, ioctl, mprotect, read SYS_292* and *write* occurred relatively more often than other events. The similar operational structures between casual game apps and arcade game apps are considered to have caused the larger occurrence of similar events.

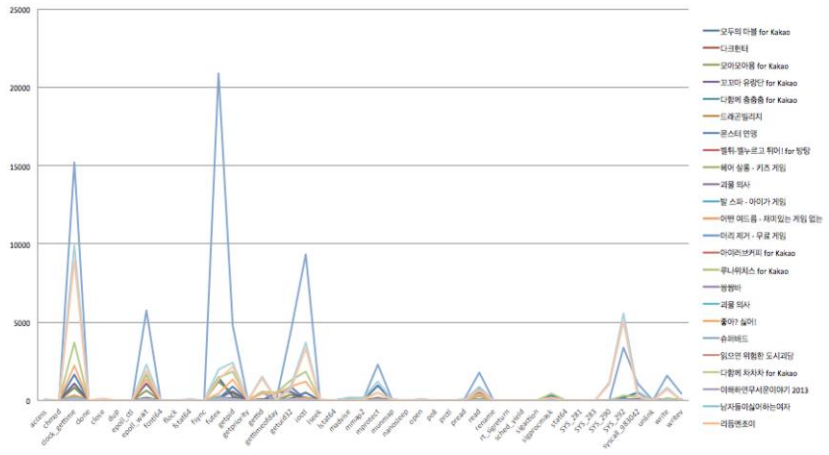

Figure 6. Normal App System Call Pattern: Casual Game

Additionally, the result analysis on event distribution of 11 normal game applications in the brain game and puzzle category was partly similar to the two categories addressed earlier, but indicated that four events of *futex*, *ioctl*, *gettimeofday* and *SYS_290* occurred relatively more often than other events. Especially applications in the brain game and puzzle category, unlike those in the arcade or casual game category, had *gettimeofday* and *SYS_290* system call events. Lastly, other than the categories mentioned earlier, event call counts of normal game application in the other categories such as the car race, card game, entertainment, sports game category can be described as follows. Distribution of system call events of normal game application in car race, sports game, entertainment and card game was similar to that of other categories, mainly showing system call events of *clock_gettime*, *epoll_wait*, *ioctl*, *getuid32* and *mprotect*.

3.2. Permission based System Call Event Analysis

Many malicious apps disguising as normal apps can be found in the official Android market, Google Play Store. This chapter will categorize malicious false permissions found in these malicious apps, and analyzed the events that occur when the normal game application with each permission is running and their characteristics.

3.2.1. Permission Categorization

As a result of analyzing permission of the most popular 80 game apps in the Google Play Store game category, there were about 20 authorizations that can damage the system, which can be categorized into four permission groups of *System*, *SMS*, *Contact* and *Location*. Therefore, permission authorizations were largely categorized into four groups, and the top 80 game apps were assigned to each group. The system category included *Maldalija for Kakao* and other 24 apps, the SMS category included Dark Hunter and other 11 apps, the contact category included *I Love Coffee for Kakao* and other 12 apps lastly and the location category included *Cookie Run* and other 7 apps.

3.2.2. App Event Analysis by Permission

Events of normal game applications included in each category were analyzed based on the result of categorizing normal game application into four permission categories [17]. Event

call counts of normal game application in the system category can be illustrated as Figure 7 below.

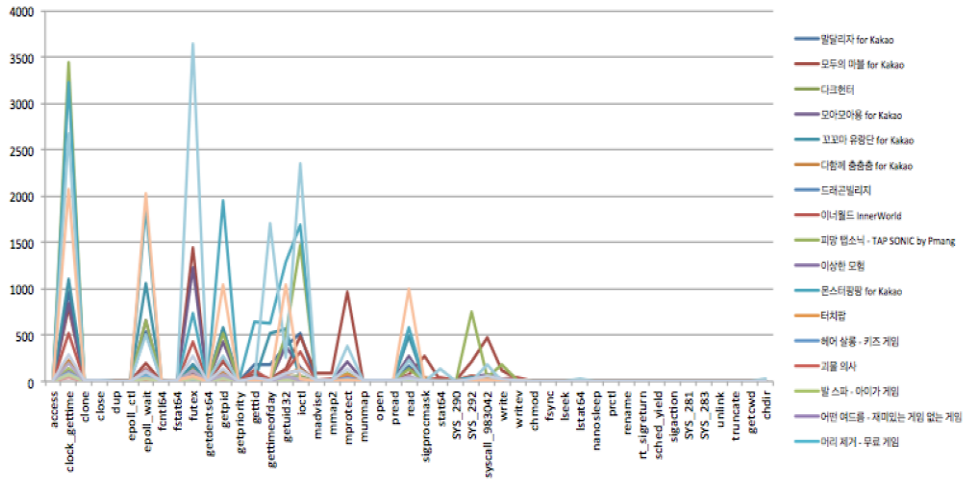

Figure 7. Normal App System Call Pattern: SYSTEM

Normal applications included in the system category showed large counts in 10 system call events of *clock_gettime, epoll_wait, futex, getpid, getuid32, ioctl, mprotect, read, SYS_292* and *syscall_983042*, and system related system call events such as *getcwd*, which is a system call event that finds recent work directory, have occurred relatively more often.

However, normal applications included in the SMS category, unlike the system category analyzed earlier, did not show large occurrence of system related system events such as *chmod, prctl* and *fysnc*. This is because they showed system call events of normal game applications that have SMS related permissions. But, the SMS related system call events including *recvmsg, recvfrom* and *send* could not be found even if normal game applications have SMS related permissions, because relevant permissions were not executed while the apps were running.

Normal applications included in the contact category did not show occurrence of system call events that occurred in the system and SMS game applications such as *lseek, chmod, rename, chdir, getcwd, truncate* and *unlink*. The reason is because the analysis was on system call event of normal game applications that have contact related permissions, but file related system call events such as message or system did not occur. However, the contact category also, as in the normal applications in the SMS category, did not show contact-related permissions while the apps were running, therefore, relevant system call could not be found.

Normal game application included in the location category also demonstrates a distribution of system call events as those in the three permission groups, which were previously analyzed. However, the graph demonstrates that the system event calls such as *close* and *open*, which were rarely occurred in the previous three permission groups, occurred in normal game applications included in the location category. A comprehensive analysis on the category and permission based analyses is as follows.

3.3. Normal Game App System Event Pattern Analysis

Normal game applications related to previous category-based analysis and permission based-analysis often have their own characteristics, but not all of them did. Therefore, this chapter suggests the characteristics of normal applications through a graph that represents all

80 events of normal game applications. Event call count graph of normal game application is as follow. Figure 9 shows the event call count of 80 normal game application, which demonstrates that *clock_gettime, clone, close, epoll_wait, futex, getpid, gettid, gettimeofday, getuid32, ioctl, madvise, mmap2, munmap, read, SYS_292, syscall_983042, write* and *writev* event can occur in normal game applications. Through this, we can assume that mainly about 18 events above occur in normal applications regularly. On the other hand, events including *dup, epoll_ctl, fcntl64, fstat64, getdents64, getpriority, open, pread, sigprocmask, stat64, SYS_290, chmod, flock, fsync, lseek, lstat64, nanosleep, poll, prctl, rename, rt_sigreturn, sched_yield, sigaction, SYS_281, SYS_283, unlink, truncate, getcwd, chdir, umask, fchown32* and others can be assumed to occur only in normal game apps or in malicious game apps.

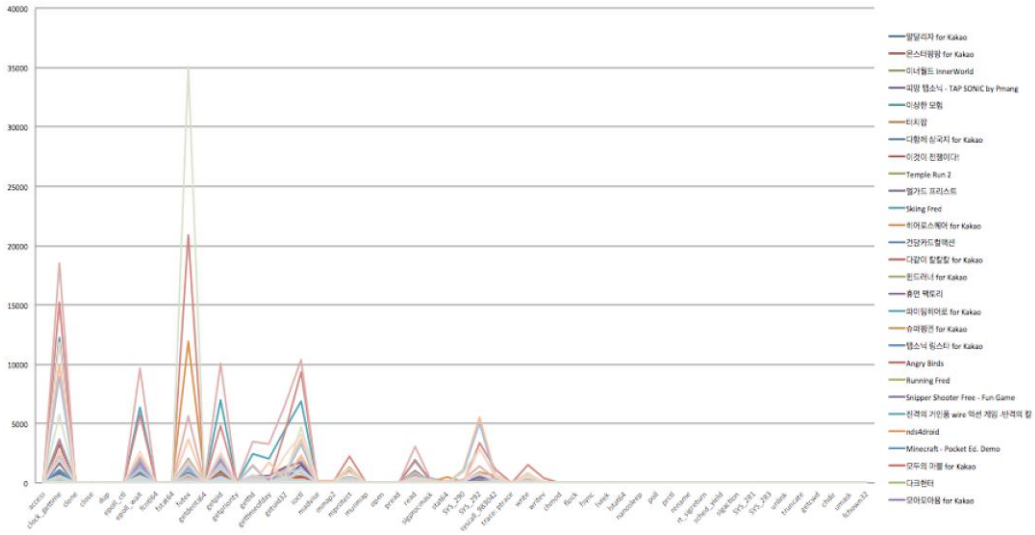

Figure 8. Normal Game App. System Call Event Pattern

4. Malicious Game App System Event Pattern Analysis

This chapter presented an analysis of 1,260 malicious samples from *Android MalGenome Project*, and then categorizes malicious apps to analyze events. Finally, we suggest characteristics of malicious application events.

4.1. Activity based Analysis

Android MalGenome Project [11] categorizes 1,260 samples of malicious apps largely by their characteristics to Malware Installation, Activation, Malicious Payloads and Permission Uses[9], and this paper analyzed Malware Installation. Malware Installation is again classified into *Repacking, Update-Attack, Drive-by Download* and *Standalone* [12, 18].

First, *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.

Secondly, *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 [19]. First method is through the user's update on existing

application. Second is a method using DEX Class Loader of Android to dynamically load compiled Android code and allowing execution of some codes of application that have not been allowed. Third is a method using Runtime of Java to dynamically load executable code that contains executable native code, or an individual file that shares a binary and is also called .so library. The last method is, like the malicious *shellcode*, dynamically loading mp3, a jpg, flash or pdf file that contains billing, and detecting vulnerabilities of external applications that access system libraries or file types to execute the attack.

The third attack, *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.

Lastly, *Standalone* is a type of app that runs by itself without help from any different tool. This Standalone can be divided to four main groups. First group distinguishes itself as spyware apps, and aims to be installed in a victim's smart phone. The second group appears to be normal, but contains false application that execute malicious action such as sending SMS without the user knowing or taking information that certifies the identification or the user. The third group contains application of malicious features such as intentionally sending unauthorized SMS or automatically registering to small additional services. The difference from the second group is that this group does not disguise as normal. Lastly, the fourth group contains applications that rely on route authorization to help it features. However, these applications use a route attack that makes a detour around the internal security sandbox without asking the user [20, 21].

4.2. Method based Analysis

Among the previous addressed four categorizations by method: *Repackaging, Update Attack, Drive-by-Download* and *Standalone*, this chapter analyzed 1260 malicious samples through *Strace* targeting the *Update-Attack* and *Drive-by Download*, which are relatively harder to detect, instead of *Repacking* and *Standalone* that are relatively easier to detect.

4.2.1. Update and Drive-by-Download Attack

Malwares contained in the *Update Attack* are *AnserverBot, BaseBridge, DroidKungFuUpdate* and *Plankton*, and each contains 187, 122, 1, and 11 apk files respectively. This paper analyzed events of malicious application targeting two of these apk files contained in these malwares.

Among the malicious apk contained in *BaseBridge*, *ArtifactDataCable* is as Figure 9 below. This malicious app changes the Wi-Fi option without the user knowing and when the app starts another application is additionally installed to damage by leaking SMS, personal information, call records or causing billings. Malwares included in *Drive-by Download* are four types of *GGTraker, JiFake, Spitmo* and *Zitmo*. This paper analyzed malicious application events targeting the apk file contained in each malware.

Battery Saver, the malicious apk in *GGTraker,* is also as Figure 9 below. This malicious app sends the phone numbers to the remote server to send premium SMS service and blocks the message that notifies this feature.

Figure 9. Update Attack (Left-BaseBridge) and Drive-by-Download Attack (Right-GGTracker)

4.2.2. System Call Pattern of Update and Drive-by-Download Attack

Among four malwares included in Update Attack, event call counts of 7 malicious applications can be illustrated as Figure 10 below.

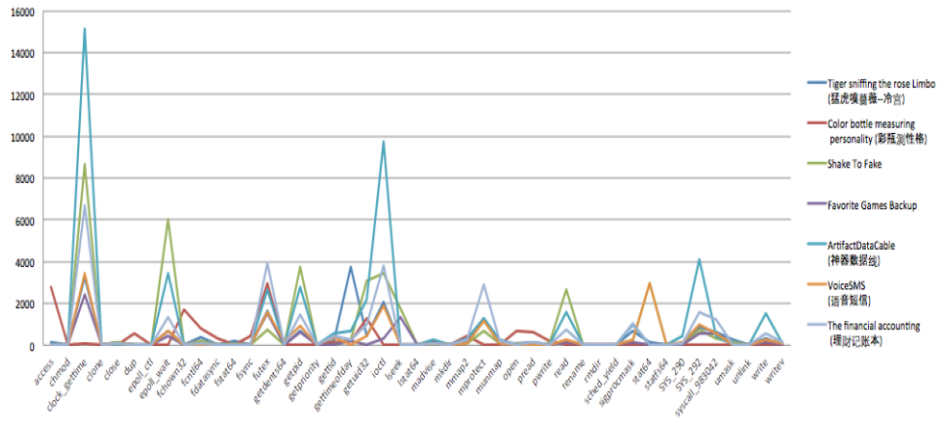

Figure 10. Malicious App. System Call Pattern – Update Attack

Analysis on event distribution targeting 7 malicious applications of Update Attack showed 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 create or delete directory or have features of synchronization of data in file disk, obtaining file system information, creation of file mask, therefore causing damage on user's device.

Among four malwares included in *Drive-by Download*, event call counts of 3 malicious applications can be illustrated as Figure 11 below. Analysis on event distribution targeting 3 malicious applications of *Drive-by Download* showed system call events of *bind*, *brk*, *connect*, *msgget*, *recv*, *recvfrom*, *select*, *semget*, *semop* and *setsockopt* which were relatively hard to be found in the earlier system call events of *Update Attack*. The occurred system call

events features 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. These features allow a supposition that *Drive-by-Download* leaks user's personal information such as SMS or financial information by executing orders from a certain external server or leads malicious application downloads by leading the user to connect to malicious URL. The next section analyzed event characteristics of 10 malicious applications analyzed in *Update Attack* and *Drive-by Download*.

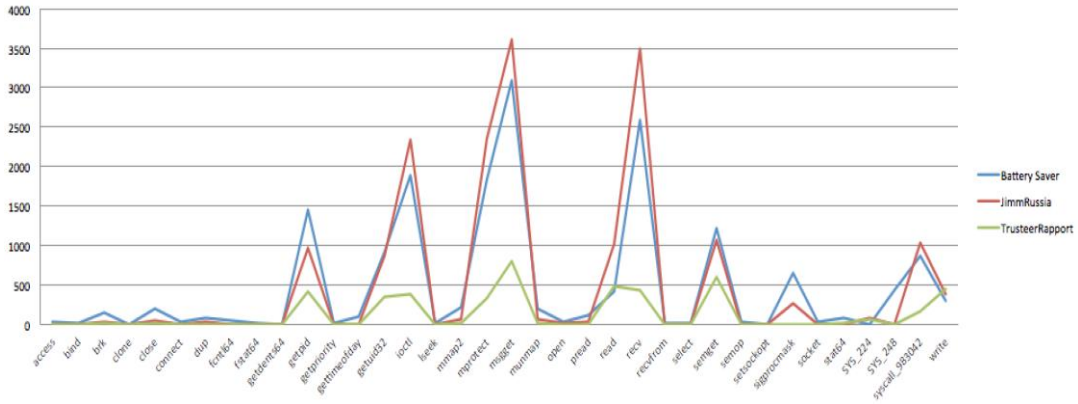

Figure 11. Malicious App. System Call Pattern – Drive-by Download

4.3. Malicious App System Event Pattern Analysis

Analysis of malicious apk contained in Update Attack and Drive-by Download addressed earlier is as follows. Figure 12 demonstrates that system call counts of *clock_gettime, epoll_wait, futex, getpid, getuid32, ioctl, mprotect, read, SYS_292, write*, which occurs mainly in normal application, are equally occurring. However, 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. Any given apps could be suspected as malicious if the 17 system call events above have occurred in the application.

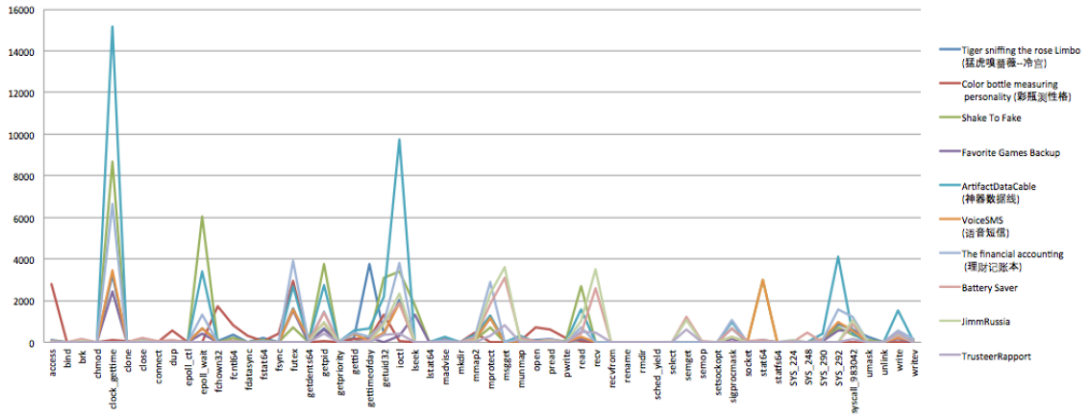

Figure 12. Malicious App, Total System Call Pattern

5. Comparison between Normal and Malicious App Event Patterns and Determination

This chapter presents a result of comparison between normal and malicious application events based on the event analysis on normal applications and malicious in chapter three and four. The analysis used a newly suggested equation to distinguish events that occur in normal and malicious applications.

5.1. System Event Pattern Comparison

The 80 normal game application and 10 malicious application events showed the events that occur only in malicious apps, those that occur only in normal apps and those that occur both in normal and malicious apps. This can be illustrated as Figure 13 (Left) below. This system calls event classification can be represented as a decision tree using formula as Figure 13 (Right).

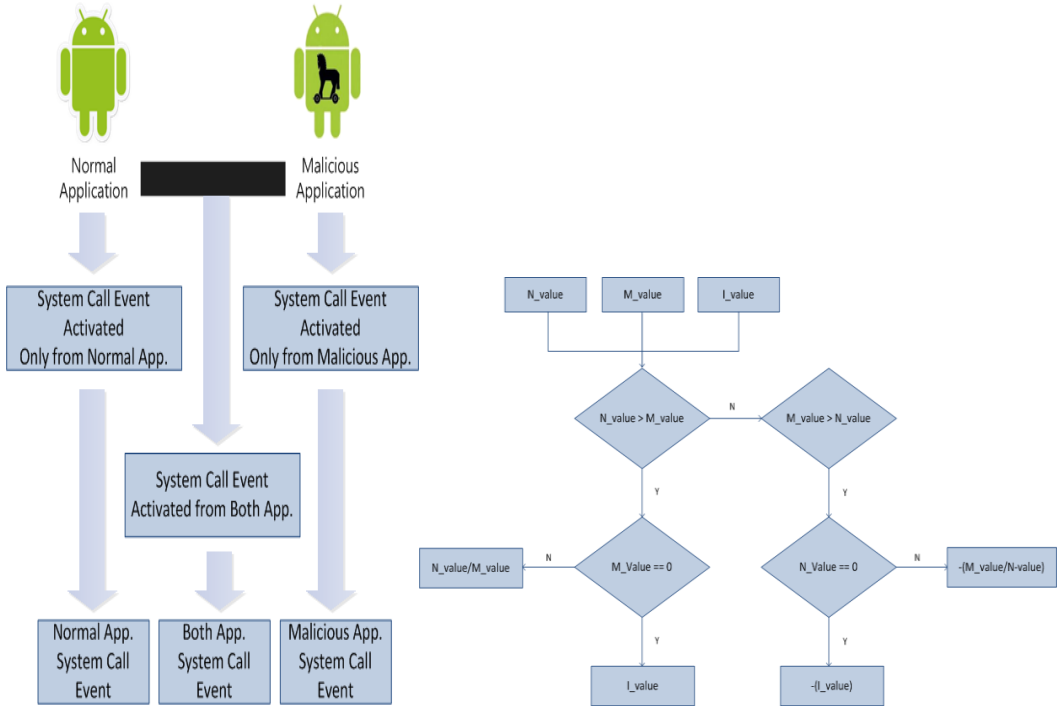

Figure 13. System Call Event Categorization (Left) and Decision Tree (Right)

Figure 14 is a graph that sets *I_value* to 50, and the system call events that occurred only in malicious application were confirmed to be 18 of *bind, brk, connect, fdatasync, mkdir, msgget, pwrite, recv, recvfrom, rename, rt_sigreturn, semget, semop, setsockopt, socket, statfs64, SYS_224* and *SYS_248*. And those that occurred only in normal application were 11 of *chdir, flock, getcwd, nanosleep, poll, prctl, rt_sigreturn, sigaction, SYS_281* and *SYS_283*. And those occurred both in normal and malicious app were 40 including *access, chmod and clock_gettime*. Next section presented a result of comparison between normal and malicious system call events while knowing if they are normal or malicious, and the next chapter will present a method that can distinguish if any found app that user installed is malicious.

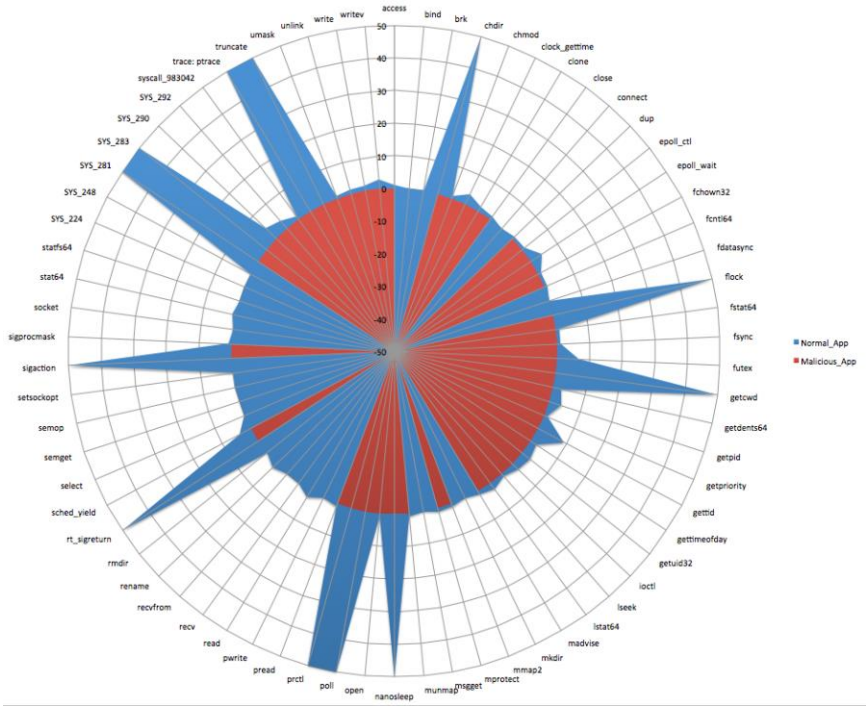

Figure 14. System Call Event Pattern Comparison

5.2. Malicious Application Distinction

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 permission. 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 permissions included in the given apps can be illustrated as below.

Similarity with normal event group and malicious event group can be drawn out by installing a found app, obtaining permissions contained in the apps and extracting events using *Strace*. Normal event group and malicious event group can be changed everytime. This paper randomly chose 5 apps that were analyzed with 32 normal event groups and 36 malicious event group, and categorized normal and malicious app events that correspond to the normal event group and normal and malicious app events that correspond to the malicious event group. Through which, S_i^{AppNml} (Similarity to Normal) and S_i^{AppMal} (Similarity to Malicious) were drawn through this and the equation, $(k\text{-}S_i^{AppNml})*(S_i^{AppMal} * k \ (i=1,2,\ldots,n)$ provides the final result of distinguishing normal or malicious. Here, S(Similarity) ranges from 0 to 1, and *k* represents the weighted value.

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 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 Figure 15 below.

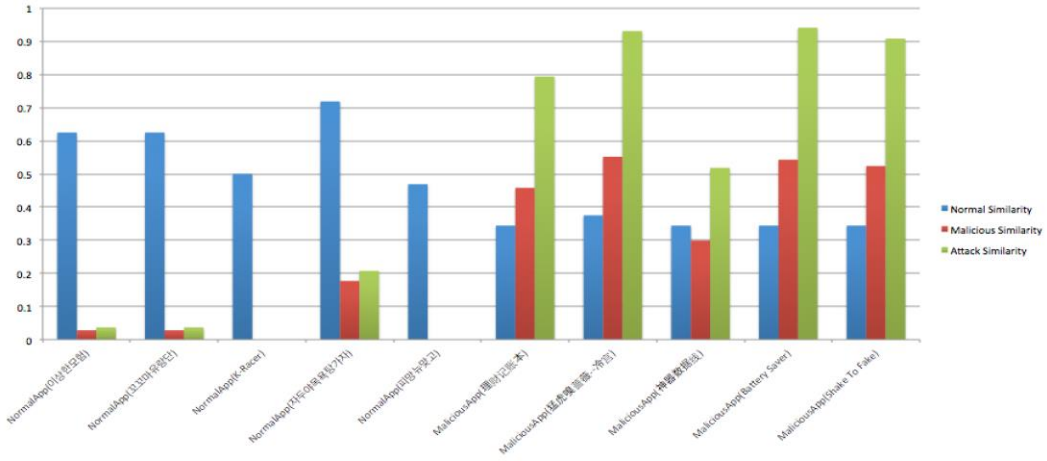

Figure 15. Normal and Malicious Similarity Calculation to Distinct Malicious Attack

S_i^{ApNml}, 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, S_i^{AppMal}, 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. Malicious app distinction similarity, obtained by a similarity calculation to make a more precise distinction on whether the events found in normal apps and those found in malicious apps are closet to malicious, can be illustrated as bellow.

In order to make a more precise distinction on whether the events found in normal apps and those found in malicious apps are closer to malicious, malicious app distinction similarity can be demonstrated by a similarity equation. The paper has suggested a malicious app distinction method on a found normal or malicious apps through by giving weighted value *k* to 14 kinds of event with malicious features in the malicious event group, including *chmod, unlink* and *fchown32* to find the similarity rate that can distinguish malicious app through the analyzed normal and malicious similarity, which have been analyzed so far. Malicious events that correspond more to the malicious event group occurred more than normal app events in Figure 15. Therefore, S_i^{AppMal}, to which the weighted value of k is multiplied, gives much larger value than S_i^{ApNml} 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.

6. Conclusions

Android-based applications can be developed in an open-source form due to the openness of Android. Therefore, apps that contain malware and disguise as a normal app can be distributed through Android official market, internet, black markets and other distribution

routes. Common mobile devices based on Android platform as well as Android devices are exposed to attack attempts by malicious apps.

This paper targeted top 80 game applications in the game category of Google Play Store and 1260 malicious samples distributed by *Android MalGenome Project* to categorize, analyzed normal and malicious app characteristics and events, and proposed an effective method for distinguishing malicious apps in Android-based common mobile device environment based on these events.

First, the forms of recent Android device risks and malicious app attacks have been analyzed. Also, the characteristics of system call events that occur in normal or malicious apps have been compared and analyzed using *Strace* module, which can collect system call events in Android kernel. Based on the result, this paper used similarity analysis algorithm on events that occurred in normal and malicious apps in Android-based common mobile devices to propose an algorithm that can distinguish malicious apps.

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. This confirmed the existence of system call function sequence of consistent pattern when an app is running.

Future research can expand the application of the method proposed in this research to more various forms of mobile apps to increase the accuracy of malware distinction on a found app, contributing to a safer user environment for common mobile devices.

Acknowledgements

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

References

[1] Google Play Android Market, https://play.google.com/store/apps.
[2] More than 700,000 malicious Android apps wreak havoc on the web, http://www.neowin.net/news/more-than-700000-malicious-apps-wreak-havoc-in-the-play-store.
[3] Android (operating system), http://en.wikipedia.org/wiki/Android_(operating_system).
[4] Uncovering Android Master Key that Makes 99% of Devices Vulnerable, 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," Proceedings of the 20th USENIX Security Symposium, **(2011)**.
[6] I. Burquera, U. Zurutuza and S. Nadjm-Tehrani, "Crowdroid: behavior-based malware detection system for Android", Proceedings of 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", 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", 2013 7th Asia Joint Conference on Information Security, **(2012)**.
[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", 2012 2nd International Conference on Computer Science and Network Technology, **(2012)**, pp. 371-375.
[10] strace, http://en.wikipedia.org/wiki/Strace
[11] Y. Zhou and X. Jiang, "Dissecting Android Malware: Characterization and Evolution," Proceedings of the 33rd IEEE Symposium on Security and Privacy, **(2012)**.
[12] X. Jiang and Y. Zhou, "Android Malware", Springer, NY, USA, **(2013)**.

[13] AhnLab V3 Mobile 2.0, http://www.ahnlab.co.kr/kr/site/product/productView.do?prodSeq=67.
[14] ALYac Android, http://asia.alyac.com/home/android.aspx.
[15] Kaspersky Internet Security for Android, http://www.kaspersky.com/android-security.
[16] BullGuard Smart Mobile Security, http://www.bullguard.com/products/bullguard-mobile-security.aspx.
[17] M. Nauman, S. Khan and X. Zhang, "Apex: Extending Android Permission Model and Enforcement with User- Defined Runtime Constraints", Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security, **(2010)**.
[18] A. PorterFelt, M. Finifter, E. Chin, S. Hanna and D. Wagner, "A Survey of Mobile Malware in the Wild", Proceedings of the 1st Workshop on Security and Privacy in Smartphones and Mobile Devices, **(2011)**.
[19] W. Zhou, Y. Zhou, X. Jiang and P. Ning, "DroidMOSS: Detecting Repackaged Smartphone Applications in Third- Party Android Marketplaces", Proceedings of the 2nd ACM Conference on Data and Application Security and Privacy, **(2012)**.
[20] D. James, "Android Game Programming For Dummies", John Wiley & Sons, Inc., Hoboken, New Jersey, USA, **(2013)**.
[21] Z. Mednieks, L. Dornin, G. Blake Meike and M. Nakamura, "Programming Android", O'Reilly, Sebastopol, CA, USA, **(2012)**.

