Android Gaming Malware Detection Using System Call Analysis

Mayank Jaiswal
Dept. of Information System Security Management
Concordia University of Edmonton
Edmonton, Canada
mjaiswal@student.concordia.ab.ca

Yasir Malik
Dept. of Information System Security Management
Concordia University of Edmonton
Edmonton, Canada
yasir.malik@concordia.ab.ca

Fehmi Jaafar
Computer Research Institute of Montreal
Dept. of Information System Security Management
Concordia University of Edmonton
Edmonton, Canada
fehmi.jaafar@crim.ca

***Abstract*—Android operating systems have become a prime target for attackers as most of the market is currently dominated by Android users. The situation gets worse when users unknowingly download or sideload cloning applications, especially gaming applications that look like benign games. In this paper, we present, a dynamic Android gaming malware detection system based on system call analysis to classify malicious and legitimate games. We performed the dynamic system call analysis on normal and malicious gaming applications while applications are in execution state. Our analysis reveals the similarities and differences between benign and malware game system calls and shows how dynamically analyzing the behavior of malicious activity through system calls during runtime makes it easier and is more effective to detect malicious applications. Experimental analysis and results shows the efficiency and effectiveness of our approach.**

***Index Terms*—Android Gaming Malware, System Call Analysis, Dynamic Analysis, Detection system, Cloning.**

I. Introduction

Android is the most popular and widely used open source mobile operating system. A great number of applications and games are offered on Google Play and other vendor stores, which have managed to dominate the market with the largest customer base [1]. The openness and customization ability allow its vendors to create a variety of games and applications, however, due to its huge popularity, Android operating systems have become a prime target for attackers to exploit devices and compromise confidentiality, integrity and availability of user's personal and financial information. According to a report, about every 11 second new malicious applications are introduced on the market and about 97% of the mobile malware are on Android [2]. The situation becomes worse, when users unknowingly download malicious clone applications (clone by either visual or behavioral similarity), from main vendor stores or by side loading, and these applications act as attacker's baits for common users to load malware onto their devices and easily execute without requiring additional permissions. Android malware has evolved its methods and functions to exploit a large number of users, and clones and repackaged malware contribute most to the total Android malware. Efforts have been made to detect the clone and repackaged malware, however, due to code obfuscation techniques, encrypted methods, UI hijacking, etc. make it difficult to detect and classify these applications as benign and malware. In Android, system calls are used to allocate hardware and system resources, and it helps applications authorized by kernel to act as an interface between applications and the kernel. The aim of our research is to analyze an Android malware application by monitoring the system calls generated during its execution state to construct the application execution behavior. We studied the execution behavior of benign and clone gaming applications, as, most of the time, gaming applications are targeted to create malicious clones. As a result, we developed a detection system that monitors the system calls of benign and malicious application along with combination of parameters and metrics (i.e. the type of system calls triggered, frequency of system calls, and their pattern, along with requested permissions) to detect the application behavior and classify it as benign and malware. The rest of the paper is organized as follows. In section II, we present the related research work on different detection and analysis techniques of Android malware. The proposed detection approach and system description is presented in section III. Section IV presents the discussion on our experiments and results. Finally, in section V, we conclude the paper, with recommendations for future work.

II. Related Work

In this section, we introduce related research work on different Android malware detection and analysis approaches. In [3] Tamer Nagy et al. describe the importance of system call analysis to construct the behavior of malware, they were able to reconstruct the behavior of the malware using system calls and detected 8 out of 10 installed malware samples using live memory analysis compare to only 4 when using memory image. In another work, Sapna et al. describe the effectiveness of dynamic analysis using the system calls to analyze the malware at runtime [4]. The researchers performed the system call analysis on 10 popular malware families and reported that in comparison to the benign applications, malicious applications generate more system calls. Franklin et al. in [5], demonstrated a scenario where user can be drawn to aid the malicious activity that is running in the background by manipulating and embedding malicious calls under any of the application UI controls. When the users knowingly or unknowingly tab on the user interface malicious system calls to the kernel are fired. The research results show the importance of deep system call analysis to detect malicious activity in such scenario.

Da et al. in [6] propose a unique malware detection method based on system call frequency to target applications that are not installed or initialized accurately. Restrictions and normalization are applied to manipulate information about system call frequency to improve the detection accuracy and training of their detection model. Similarly, Mathura et al. in [7] presented the comparison of system calls made by applications in two different modes (post installation, and during malicious activity) to show the similarities and dissimilarities of system call in two different modes. Marko Dimjasevic et al. in [8] propose a new technique that performs the automatic malware classification by tracking system calls while the applications are executed in a sandbox environment. The technique includes three phases: dynamic analysis, feature extraction, and machine learning. In this research, 96% detection accuracy is reported using different machine learning algorithm implementations.

Burguera et al. describe the program execution behavior and state that while a program is in user mode, it does not have access to kernel mode to access RAM other hardware resources. Thus, it relies on system calls and requests all services from the operating system. The authors propose a system call-based model to detect an Android malicious application; however, this approach requires huge resource utilization and large computation to detect and compare the system calls [9].

In another work, Canfora et al. in [10] propose an Android malware detection method based on sequences of system calls that addresses the main limitations of existing methods that include low accuracy, proneness to evasion techniques, and weak validation, often limited to emulators or modified kernels. Their system shows improved detection accuracy with low computational requirements. Das et al. in [11] argue that mapping the system call to functional behavior of the application is not sufficient They analyzed 534 mobile applications and concluded that using only system calls as features is not sufficient when trying to carry out application behavioral classification.

III. Gaming Malware Detection System

In this section, we present, a dynamic Android gaming malware detection system based on system call analysis to differentiate between the malicious and legitimate games. Our goal here is to develop an automated detection system for Android malware behavior based on system call analysis. This experimental observation is based dynamic analysis of malware during the development of a detection system. All normal applications and malware have different system calls patterns, i.e. every application generates a system call according to its behavior (for instance, `read()`, `unmask()`, `write()`, etc.). While analyzing the system call patterns of benign and clone malware game application, we observe that there are some system calls that are used by both benign and malware applications which are difficult to detect, however, the they exhibit different system call frequency which can help in classify the app as benign or malicious.

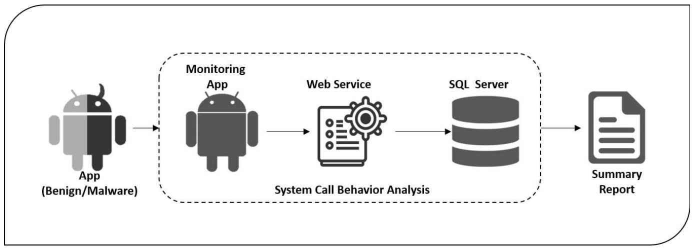
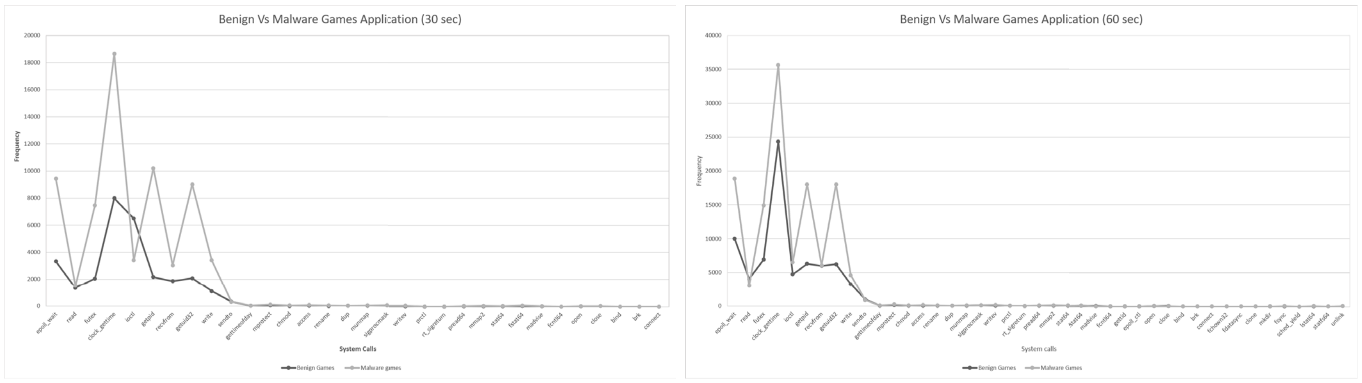


Fig. 1: Gaming Malware Detection System

In addition, clone application request high level permission from the user and compromise the system by exploiting the granted permission, which leads to consuming more resources or performing unnecessary operations e.g. CURD operations. In this work, we developed an Android detection system to record system call events and using a combination of parameters and metrics (For instance, the type of system calls triggered, frequency of system calls, and their pattern, along with requested permissions) to detect the application behavior. Fig. 1 shows the system diagram of the proposed detection system. When any new application is installed, the monitoring application detects the application and performs STRACE command on it to generate a system call summary that includes (application id, system call name, frequency and call time). Our detection system includes a monitoring application which is installed on a Memu emulator and runs continuously in the background with super user rights to capture the system call events from malicious and benign game applications. When an application is installed, the monitoring application detects it and adds an application package name to “realm” database and add a new application with the package name of the sample application to the database. Once the system is trained with a complete data set, whenever any new (benign or malware) game application is installed, the monitoring application detects it, analyze the system call event and make decision on application behavior.

IV. Experimental Results and Discussion

In this section, we present the assessment of our detection system by testing malicious and normal game applications samples. The detection system is trained by performing numerous experiments, we performed a system call analysis of 20 normal game applications and 40 malicious game applications in different settings and Android versions. To analyze the system calls of normal games, 20 free games applications were downloaded from Google Play store. These games are then executed in our system to record the



(a) 30 seconds

(b) 60 seconds

Fig. 2: System Call Pattern of Benign Vs Malware Game Applications

system calls pattern and frequencies during runtime. During the experiments, we observed, that some system calls like `clock_gettime`, `epoll_wait`, `futex`, `getpid`, `getuid32`, `ioct` and `read` occurred more frequently compare to other system calls. Similarly, to analyze the system calls of malware games, malware game applications samples are collected from the virus total data repository. When these samples were executed in our system, some major differences were recorded in both the system calls events and its frequencies. During the experiments, system calls such as `brk`, `bind`, `fchown32`, `fdatasync`, `setsockopt`, `getsocket`, `gettimeofday` were mostly found in malware game applications. We also observed that the system calls `epoll_wait`, `getuid`, `getpid`, `clock_gettime` are found with high frequency in malware games.

Initially, we recorded system calls for the first 30 seconds for benign and malicious applications, but neither application showed any overt behavior in system call frequency and pattern. Later, we repeated the experiment for the first 60 seconds to get some comparable results. For most of the malicious games, the highest frequency for system call ``` ``clock_gettime'' ``` was 35000, while the highest frequency for benign games was 25000. System calls such as `brk`, `bind`, `fchown32`, `fdatasync`, `setsockopt`, `getsocket gettimeofday` showed a higher frequency in gaming malware games comparable to benign games. The comparison of system calls events for benign and malware games for 30 seconds and 60 seconds are shown in Fig. a and b respectively. This information about frequency along with other features listed above are used to detect and classify benign and malicious gaming applications.

We also tested the clone malware applications to observe its system call behavior. For the purpose of testing, we ran ``` ``bus parking 3d'' ``` application from Google store and we found more than 20 applications with the same name and in some cases similar icons from different developers. Initially, we executed the benign bus parking application and recorded, the system call for 30 and 60 Sec. Later, we ran the clone version of the same application and recorded the system calls for the same periods. During multiple experiments, we noted the frequency of the system call events such as futex for the clone game appear higher in clone compare to benign game (60 times in the benign game compared to 5461 times in the clone game). The details of the system call operation and frequency for the clone and original games are shown in Table I, which shows the results, the high number of system calls degrades the device performance and halt other applications. Along with common system calls, some additional system calls such as `brk`, `llseek`, `stat64`, `pread64` calls were also generated by the clone game which were not appeared in benign game.

While executing the malicious games, we also observed that most of them required device administrator permissions to gain access to the device. TableII shows the Android application permissions requested by the malicious clone and legitimate application. It clearly shows that the malicious clone application request for many permissions which can lead to malicious activities such as stealing user identity, sending GPS location, taking control over the infected phone, killing other application processes and so on.

V. Conclusion

There are a number of research has been done using Android system call analysis and proven that it is a powerful technique. Android malware has evolved its methods and functions and clones and repackaged malware contribute most to the total Android malware and are difficult to detect. In this research, we develop an Android malware gaming detection system based on application system call analysis during its execution state. Our approach does not require source code of the Android application to analyze application behavior, and can be easily be adopted and implemented in the monitoring application. We tested out system with clone gaming applications to assess the effectiveness and efficiency of our approach. A system call analysis of Android gaming malware was performed, and their behavior was categorized depending on their similarities and dissimilarities. The results

TABLE I: List of System Calls in Benign and Malicious Clone Application

Operations	Number of System Calls	
	Benign App	***Malicious Clone App***
epoll_wait	2613	4703
clock_gettime	3796	17251
read	1369	3179
write	363	3913
futex	60	5461
getpid	1349	2688
ioctl	543	5803
getuid32	1336	2604
recvfrom	524	5149
mprotect	22	155
sendto	46	367
chmod	1	2
access	3	8
rename	1	2
dup	5	20
gettimeofday	3	39
munmap	5	43
sigprocmask	3	82
writev	3	26
prctl	2	9
rt_sigreturn	1	1
mmap2	5	45
fstat64	14	140
Madvise	5	30
fcntl64	2	8
Gettid	2	9
epoll_ctl	2	8
Open	7	72
Close	17	111
stat64		13
pread64		22
Brk		1
_llseek		6

show that some system call events were observed in malicious applications and can be used to create a signature to detect malware. In addition, while analyzing clone applications, we found that they request a high-level permission from users and compromise the system by exploiting the granted permission. In the future, we will extend our work to include other Android malware families and application features to generalize the results and improve its efficiency.

ACKNOWLEDGMENT

This work has been partly funded by a Concordia University of Edmonton, Public Safety Canada grant, and by the Computer Research Institute of Montréal (CRIM). We gratefully thank the Ministère de l'Économie, Science et Innovation du Québec (MESI) for their generous support.

REFERENCES

[1] A. B. Ayed, "A literature review on android permission system," *International Journal of Advanced Research in Computer Engineering and Technology.*, vol. 4, no. 4, 2014.

[2] L. Sun, Z. Li, Q. Yan, W. Srisa-an, and Y. Pan, "Sigpid: significant permission identification for android malware detection," in *Malicious and Unwanted Software (MALWARE), 2016 11th International Conference on*, pp. 1–8, IEEE, 2016.

TABLE II: List of Permissions in Benign and Malicious Clone Application

Permissions Requested	
Benign Game	***Malicious Clone Game***
android.permission. READ_PHONE_STATE *(read phone state and identity)*	android.permission. ACCESS_FINE_LOCATION *(fine GPS location)*
android.permission. ACCESS_WIFI_STATE *(view Wi-Fi status)*	android. permission. WRITE_APN_SETTINGS *(write access point name settings)*
android.permission. ACCESS_NETWORK_STATE *(view network status)*	android. permission. RECEIVE_BOOT_COMPLETED *(automatically start at boot)*
android.permission. INTERNET (full Internet access)	android. permission. INTERNET *(full Internet access)*
android.permission. WRITE_EXTERNAL_STORAGE *(modify/delete SD card contents)*	com. android.launcher.permission. INSTALL_SHORTCUT *(Unknown permission from android reference)*
com.android.vending.BILLING *(unknown permission from Android reference)*	android. permission. SYSTEM_ALERT_WINDOW *(display system-level alerts)*
	android. permission. READ_LOGS *(read sensitive log data)*
	com. google. android. c2dm.permission. RECEIVE *(unknown permission from android reference)*
	android. permission. ACCESS_NETWORK_STATE *(view network status)*
	android. permission. ACCESS_COARSE_LOCATION *(coarse (network-based) location)*
	android. permission. WAKE_LOCK *(prevent phone from sleeping)*
	android.permission.GET_TASKS *(retrieve running applications)*
	com.tapinator.bus.parking. permission.C2D _MESSAGE *(C2DM permission)*
	android. permission. ACCESS_WIFI_STATE *(view Wi-Fi status)*
	android. permission. READ_PHONE_STATE *(read phone state and identity)*
	android. permission. RESTART_PACKAGES *(kill background processes)*
	android. permission. WRITE_EXTERNAL_STORAGE *(modify/delete SD card contents)*
	com. android. vending. BILLING *(unknown permission from android reference)*
	android.permission. GET_ACCOUNTS *(discover known accounts)*

[3] P. Z. Tamer Nagy, Dale Lindskog, "Analytic comparison between live memory analysis and memory image analysis in android environment," in *Proceedings of the 2nd World Congress on Computer Applications and Information Systems (WCCAIS'2015)*, 2015.

[4] K. K. Sapna Malik, "System call analysis of android malware families," *Indian Journal of Science and Technology.*, vol. 9, no. 21, 2016.

[5] P. Tchakounte Franklin, Dayang, "System calls analysis of malwares on android," *Maejo International Journal of Science and Technology*,

vol. 2, no. 9, 2013.

[6] C. Da, Z. Hongmei, and Z. Xiangli, “Detection of android malware security on system calls,” in *2016 IEEE Advanced Information Management, Communicates, Electronic and Automation Control Conference (IMCEC)*, pp. 974–978, Oct 2016.

[7] S. K. Mathura Bai B, “Malware analysis using system call monitoring,” *International Journal of Electrical, Electronics and Computer Systems (IJEECS)*, 2015.

[8] M. Dimjašević, S. Atzeni, I. Ugrina, and Z. Rakamaric, “Evaluation of android malware detection based on system calls,” in *Proceedings of the 2016 ACM on International Workshop on Security And Privacy Analytics*, IWSPA ’16, (New York, NY, USA), pp. 1–8, ACM, 2016.

[9] I. Burguera, U. Zurutuza, and S. Nadjm-Tehrani, “Crowdroid: behavior-based malware detection system for android,” in *Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices*, pp. 15–26, ACM, 2011.

[10] G. Canfora, E. Medvet, F. Mercaldo, and C. A. Visaggio, “Detecting android malware using sequences of system calls,” in *Proceedings of the 3rd International Workshop on Software Development Lifecycle for Mobile*, DeMobile 2015, (New York, NY, USA), pp. 13–20, ACM, 2015.

[11] P. K. Das, A. Joshi, and T. Finin, “App behavioral analysis using system calls,” in *2017 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pp. 487–492, May 2017.