A Design of the Event Trigger for Android Application

Ting Hu, Zhuo Ning, and Zhixin Sun(✉)

Key Laboratory of Broadband Wireless Communication and Sensor Network Technology, Nanjing University of Posts and Telecommunications, Nanjing, China
sunzx@njupt.edu.cn

Abstract. The exploding of Android malware makes security analysis more important and urgently calls for automation in its analysis. Often automation analysis includes static and dynamic methods. And an important work of the dynamic analysis is gathering accurate behavior information of Android apps. However, traditional methods, which are used to inject random events to exercise the user interface, can not capture the behavior triggered by the event. To overcome the above shortcomings, this paper designs a framework of the Android malware detection based on cloud and focuses on how to design an Event Trigger to trigger more behaviors. This method can enlarge the dynamic analysis scope to find more information of malicious behaviors.

Keywords: Android malware · Dynamic analysis · Event Trigger

1 Introduction

In recent years, smart phone sales have grown tremendously. Until the first quarter of 2015, according to the report from Gartner, worldwide sales of smart phones to end users have reached 336 million [1]. Among various platforms, Google's smart phone platform, Android, has captured more than 75% of the total market-share. In another word, it is the most popular operating system at present. Unfortunately this explosive growth also has drawn the attention of cyber criminals who try to trick the user into installing malicious software on the device.

Android terminal stores a lot of personal information, such as contacts, messages, social network access, browsing history and banking credentials, so it has become a prime target for malicious attacks. Android malwares such as premium rate SMS Trojans, spyware, botnet, aggressive adware and privilege escalation attack have reported exponential rise from the Google Play store and well known third-party market places. According to the statistics from Kasper sky, the number of malicious Android applications topped the 10 million mark in January 2014 [2]. These malicious applications pose a great security risk to mobile phone owners and solving the security issue of Android has become a hot topic in the field of information security.

Given the enormous growth of Android malware, security researchers and vendors must analyze more and more applications (apps) in a given period of time to understand the purpose of the software and to develop countermeasures accordingly. Through the

J.-H. Lee and S. Pack (Eds.): QShine 2016, LNICST 199, pp. 426–434, 2017.
DOI: 10.1007/978-3-319-60717-7_42

efforts of researchers, Android security has made much progress both in static and dynamic analysis [3]. The classical approach to automated analysis of suspicious applications is static analysis. Static analysis investigates software properties that can only be investigated by inspecting the downloaded app and its source code [4]. A typical example of it is signature based detection similar to the common approach of antivirus [5]. However, malware usually uses obfuscation techniques to puzzle static analysis [6, 7]. Thus dynamic analysis does not inspect the source code, but rather executes it within a controlled environment, often called sandbox to get a more accurate result. By monitoring and logging every relevant operation of the execution (such as sending SMS messages, reading data from storage, and connecting to remote servers), an analysis report is automatically generated. Dynamic analysis can combat obfuscation techniques rather well, but it is thwarted by runtime detection methods. Therefore, combing the static and the dynamic usually makes sense in practice.

A big problem of dynamic analysis is how to trigger malicious behaviors as many as it can, especially for those one which could not be triggered just by installing the application. And some behaviors can not be triggered except for particular interaction. Generally, for each Activity, Monkey [8] is used to inject random UI events to exercise the user interface. Furthermore, some behaviors of Android apps are triggered by events, such as the arrival of new SMS and location change, which sometimes could not be triggered by random event streams. Our Event Trigger can inject in-time fake events at the most appropriate execution time, which can enlarge the scope for analyzing Android apps and make the report more accurate.

Last but not least, static analysis, dynamic analysis and other processes of our research require a large amount of computing resource. Therefore, we deploy the core solution on the cloud to provide the parallel service for a large number of smart phones.

2 Background

In order to automatically install the application and simulate the user operation in the actual equipment or simulator, most of the detection schemes used automatic control scripts. For instance, TaintDroid [9], DroidBox [10], AppPlayground [11], PuppetDroid [12], Andrubis [13] and Mobile-Sandbox [14] used the MonkeyRunner provided by Android SDK, which could generate enough random User Interface (UI) events to ensure a large number of interactive behaviors are triggered. However, the event frequency of different applications varies considerably, and the random UI events can not target the application's vulnerability. DroidTrace solved the problem in a different way. It triggered different dynamic loading behaviors by physical modification. Firstly, it found the function which loaded other functions dynamically; then inserted the trigger code into these functions to generate forward execution path; finally packaged this app as a new application [15]. In this way, DroidTrace can trigger targeted behaviors, but it was not a real-time solution. Both of the above two methods can not trigger behaviors which can only be triggered by particular events. For example, if the user moved to a new place, the location will be changed, and such location change event cannot be achieved by both DroidTrace and automatic control script. Another example of the event trigger is receiving short messages and calls. All

these events could trigger some malicious behaviors of application, such as leaking the location change or the received new SMS.

To overcome the above shortcomings, an event trigger is explored. It injects recurrent fake events, including the arrival of new SMS, calls and location changing, to trigger the malicious behaviors as many as possible during the execution. Compared with the ordinary, it can perform in-time event injection at the most appropriate time, for not only callbacks listed in the Manifest but also API callbacks that invoked at runtime. At the same time, it provides more behavior analysis information for the developers or users to promote the accuracy.

Currently, there is a tendency to malware detection service from the host terminal to the cloud. Cloud computing is the development product of distributed computing, parallel computing and utility computing. It congregates large numbers of computation resources and provides on-demand IT services to the remote Internet users. Wang et al. [16] provided a new android multimedia framework based on Gstreamer. It can greatly improve the multimedia processing ability in terms of efficiency, compatibility, feasibility and universality. Cloud resources and their loads possess dynamic characteristics. Zuo et al. [17] proposed a scheduling method called interlacing peak which can balance loads and improve the effects of resource allocation and utilization effectively. Meanwhile they proposed a Self adaptive threshold based Dynamically Weighted load evaluation Method (termed SDWM) [18]. It evaluates the load state of the resource through a dynamically weighted evaluation method. For task-scheduling problems in cloud computing, a multi-objective optimization method is also proposed [19]. In this paper, we deploy the core solution on the cloud to provide the parallel service.

The paper is organized as the following. The second section discusses the implement of it in details, including the process design, the communication architecture and the function module. In the third section a security detection architecture based on cloud is designed accordingly.

3 Event Trigger Implementation

3.1 Process of the Event Trigger

The Event Trigger runs in the sandbox when faking events, while its specific process is shown in Fig. 1.

Step 1: Firstly, the application manifest file (AndroidManifest.xml) will be analyzed to extract API list of registered callback functions, which are predefined event functions;
Step 2: Running an application in the sandbox and notifying the Event Trigger to inject related events when a callback function is registered through the API;
Step 3: The Event Trigger injects related fake events, and then the hardware device will detect the occurrence of the event and notify the application to update the latest information.

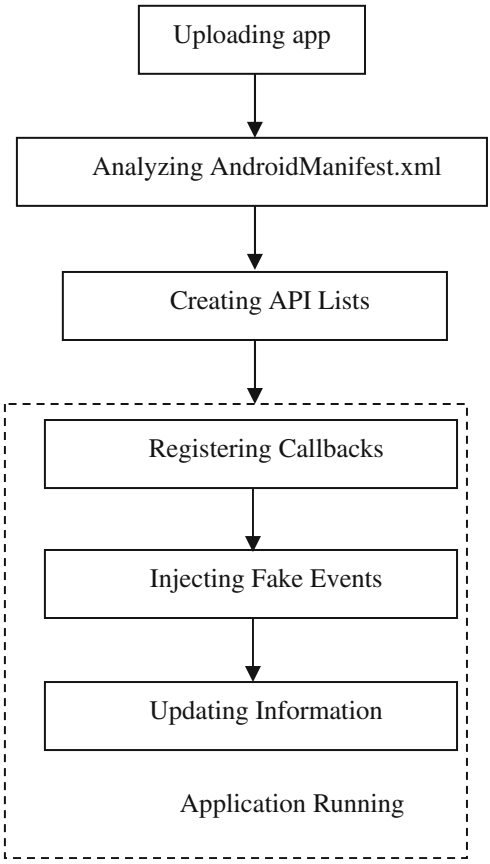

Fig. 1. Process of the Event Trigger

3.2 Communication Architecture of the Event Trigger

The class that implements the service in the Android framework layer is ManagerService. ServiceManager is another class dedicated to the management of system services. And it is responsible for the registration and management of all system services. Both of them communicate via the Binder protocol. When the application calls for a system service, it needs to invoke the service through the service agent and then sends a request to the system server process via the inter process communication, then the process is responsible to return the results. The entire communication architecture is shown in Fig. 2. For example, the Event Trigger will automatically inject fake location change events when the corresponding callbacks are registered through the LocationManager._requestLocationUpdates() API. Then the location change monitoring request is delegated to LocationManagerService via the Binder protocol, which will notify LocationManagerService immediately to fake a location change event. Finally, the hardware will detect an event occurred and notice the application to update the

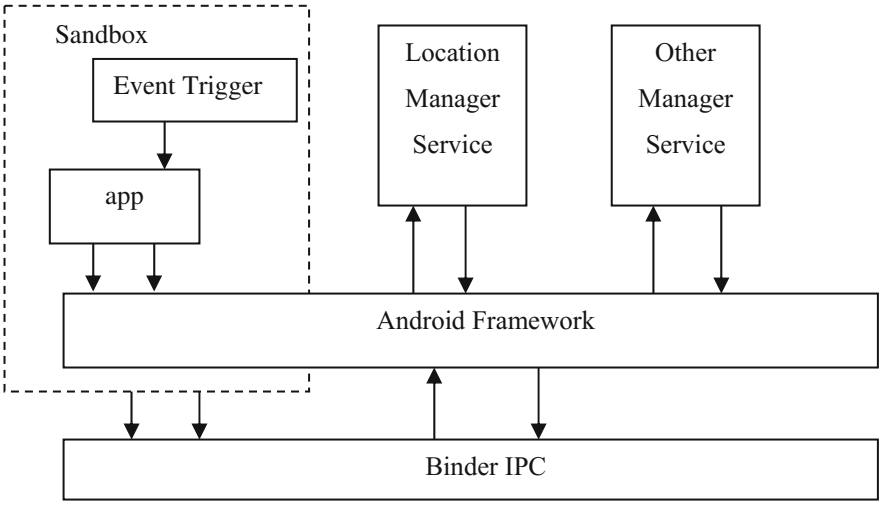

Fig. 2. Framework of the Event Trigger in Android

latest information. Thus, the registered callback could be triggered for execution to enlarge the dynamic analysis scope to find more information of malicious behavior.

3.3 Function Module of the Event Trigger

The Event Trigger includes a manifest parsing module, an event pre-triggering module, an event faking module, and an event triggering module, as shown in Fig. 3.

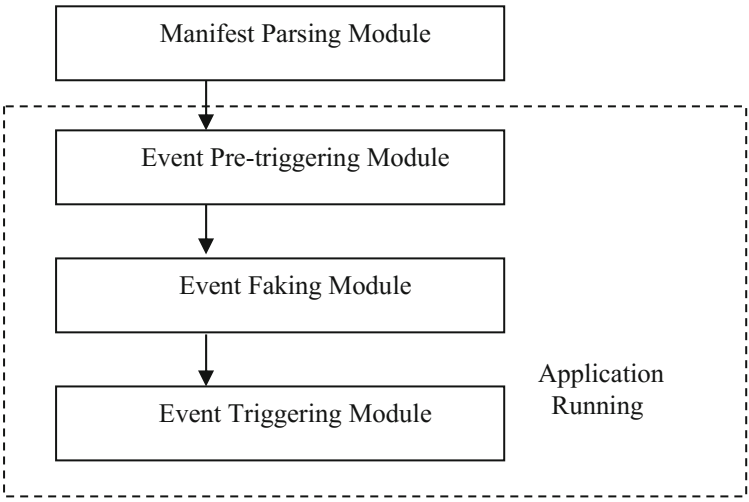

Fig. 3. Function module diagram of the Event Trigger

Manifest Parsing Module, which parses the application manifest file (AndroidManifest.xml) to extract API list of registered callback functions, namely the predefined some event functions.

Event Pre-triggering Module, which will send a request to the Event Faking Module to inject corresponding false events when an application wants to use a system service, the callbacks of which will be registered through the API, that is, some of the predefined event functions will soon be called.

Event Faking Module, which could perform in-time event injection when received the request information for fake events. The fake events include not only callbacks listed in the Manifest but also API callbacks that invoked at runtime.

Event Triggering Module, which could trigger fake events and the hardware will sniff an event occurred and notice the application to update the latest information. Thus, the registered callback could be triggered for execution to enlarge the dynamic analysis scope to find more information of malicious behavior.

The fake events include location change, the arrival of new SMS, receiving calls, etc. in the Event Faking Module. Take the location change injection as an example, the whole process of which is running in the process of the sandbox. The Event Faking Module could firstly fake current geographic coordinates (Android simulator in sandbox cannot automatically position coordinates) when received the request information for location injection from the Event Pre-triggering Module. Secondly, a series of location information initialization: location initialization, interface initialization, search module initialization, data initialization and so on, are prepared for the location change. Then run the thread to set the coordinate information of the location change. Finally, destroy the geographic coordinate information after the location change event is triggered and wait for the next trigger.

4 The Security Detection Architecture for Android Application

In this paper, we intend to develop an Android malware detection system, which includes the client and the server with a synergy framework based on cloud.

The Client. Prototype system installs itself as the Android app on the user's smart phone to detect the specific software. Users can upload the Android APK file to the cloud manually.

The Cloud. The static analysis and the dynamic analysis of our research require a large amount of computing resource. Therefore, we deploy the core solution on the cloud to provide the parallel service for a large number of smart phones. We suggest that each end host run a lightweight process to acquire executables entering a system, send them into the network for analysis, and then run or quarantine them based on a threat report returned by the network service. The system monitors the dangerous functions. When sufficient evidences are found, the request/response manager will upload the suspicious applications. The cloud server is responsible to detect the vulnerability and maliciousness of the uploaded app. After the completion of detection, it will return back the evaluation information. Finally, the App manager will determine the next steps to be

taken. The Event Trigger runs in the sandbox when faking events. The architecture of the synergy framework is shown in Fig. 4.

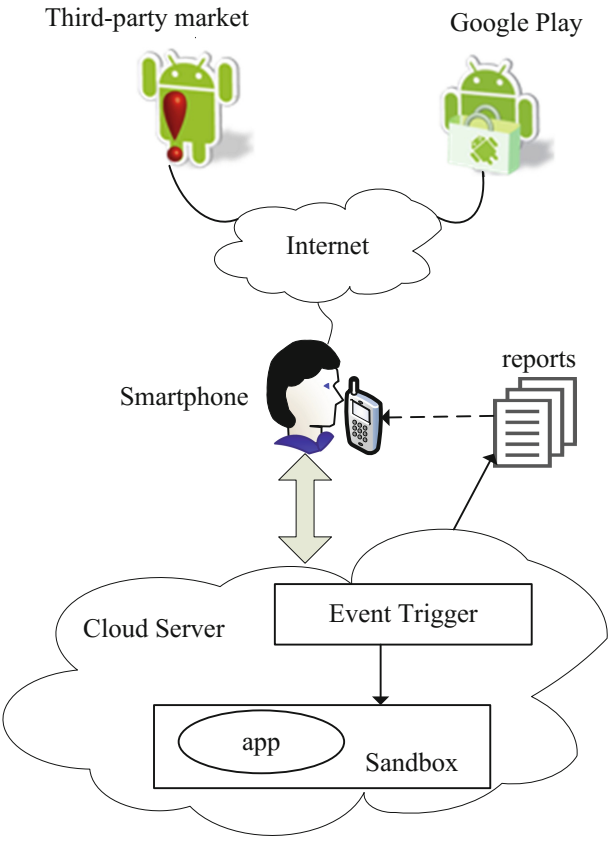

Fig. 4. Deployment of the security synergy detection framework.

This paper uses open source cloud computing platform Eucalyptus established by Santa Barbara universities to achieve specific architecture [20]. And we use virtual machine nodes in the cloud to implement parallel distributed detection with dynamic and static analysis. Meanwhile, the enclosed environment in virtual machine nodes is used to construct the corresponding event triggers to monitor dynamic behavior in the sandbox. The distributed Propagation of Information with Feedback (PIF) protocol algorithm is used to formally describe the procedure of dynamic analysis and analysis report return.

Our experiment results show that in many cases we can: detect the existence of trigger-based behavior, find the conditions that trigger such hidden behavior, and find inputs that satisfy those conditions and advance its performance.

5 Conclusion

In this paper, we propose an Event Trigger which can automatically trigger the application event behaviors. Compared with existing event injection techniques, our technique could perform in-time event injection at the most appropriate time, for not only callbacks listed in the Manifest but also API callbacks that invoked at runtime. With the help of runtime injected events, the scope of the dynamic analysis is enlarged, and the accuracy is also promoted.

Though many techniques are introduced to drive the application execution, it is worth noting that our Event Trigger could not guarantee a complete coverage over all possible behaviors. Generally it is a difficult problem for all dynamic analysis work. This paper tries to design a better behavior approximation for analyzing Android apps, and leaves the coverage problem as our future work.

Acknowledgments. This paper was supported by the National Natural Science Foundation of China (Nos. 61170276, 61373135) and the Key University Science Research Project of Jiangsu Province (Grant No. 12KJA520003).

References

1. Gartner, Inc.: Gartner Says Emerging Markets Drove Worldwide Smartphone Sales to 19 Percent Growth in First Quarter of 2015. http://www.gartner.com/newsroom/id/3061917
2. Kaspersky, Lab.: Number-of-the-week-list-of-malicious-Android-apps-hits-10-million (2014). http://www.kaspersky.com/about/news/virus/2014/Number-of-the-week-list-of-malicious-Android-apps-hits-10-millon
3. Enck, W.: Defending users against smartphone apps: techniques and future directions. In: Jajodia, S., Mazumdar, C. (eds.) ICISS 2011. LNCS, vol. 7093, pp. 49–70. Springer, Heidelberg (2011). doi:10.1007/978-3-642-25560-1_3
4. Application Exerciser Monkey. http://developer.android.com/tools/help/monkey
5. Schmeelk, S., Yang J., Aho, A.: Android malware static analysis techniques. In: 10th Annual Cyber and Information Security Research Conference, pp. 1–2. ACM, New York (2015)
6. Batyuk, L., Herpich, M.: Using static analysis for automatic assessment and mitigation of unwanted and malicious activities within Android applications. In: Malicious and Unwanted Software (MALWARE), pp. 66–72. IEEE, Fajardo (2011)
7. Moser, A., Kruegel, C., Kirda, E.: Limits of static analysis for malware detection. In: Proceedings of the 23rd Annual Computer Security Applications Conference (2007)
8. Willems, C., Freiling, F.C.: Reverse code engineering—state of the art and countermeasures. it-Inf. Technol. 53–63 (2011)
9. Enck, W., Gilbert, P., Chun, B.-G., Cox, L. P., Jung, J., McDaniel, P., Sheth, A.N.: TaintDroid: an information-flow tracking system for realtime privacy monitoring on smartphones. In: Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (OSDI), October 2010
10. Dynamic analysis of Android apps. https://github.com/pjlantz/droidbox
11. Rastogi, V., Chen, Y., Enck, W.: Appsplayground: automatic security analysis of smartphone applications. In: Proceedings of the Third ACM Conference on Data and Application Security and Privacy, CODASPY 2013. ACM, New York (2013)

12. Gianazza, A., Maggi, F., Fattori, A., Cavallaro, L., Zanero, S.: PuppetDroid: a user-centric UI exerciser for automatic dynamic analysis of similar android applications. CoRR, vol. abs/1402.4826, 2014
13. Lindorfer, M., Neugschwandtner, M., Weichselbaum, L., Fratantonio, Y., van der Veen, V., Platzer, C.: Andrubis-1,000,000 apps later: a view on current Android malware behaviors. In: Proceedings of the 3rd International Workshop on Building Analysis Datasets and Gathering Experience Returns for Security, BADGERS (2014)
14. Spreitzenbarth, M., Schreck, T., Echtler, F., Arp, D., Hoffmann, J.: Mobile-Sandbox: combining static and dynamic analysis with machine-learning techniques. Int. J. Inf. Secur. **14**(2), 141–153 (2015). Springer, Berlin
15. Zheng, M., Sun, M., Lui, J.C.S.: DroidTrace: a ptrace based Android dynamic analysis system with forward execution capability. In: Wireless Communications and Mobile Computing Conference (IWCMC), pp. 128–133. IEEE Press, Nicosia (2014)
16. Wang, H., Hao, F., Zhu, C., Rodrigues, J.J.P.C., Yang, L.T.: An android multimedia framework based on Gstreamer. In: Rodrigues, J.J.P.C., Zhou, L., Chen, M., Kailas, A. (eds.) GreeNets 2011. LNICSSITE, vol. 51, pp. 51–62. Springer, Heidelberg (2012). doi:10.1007/978-3-642-33368-2_5
17. Zuo, L., Shu, L., Dong, S., Zhu, C., Han, G.: A multi-queue interlacing peak scheduling method based on tasks classification in cloud computing. IEEE Syst. J. (2016)
18. Zuo, L., Shu, L., Dong, S., Zhu, C., Zhou, Z.: Dynamic weighted load evaluation model based on self-adaptive threshold in cloud computing. ACM Mob. Netw. Appl. 1–15 (2016)
19. Zuo, L., Shu, L., Dong, S., Zhu, C., Hara, T.: A multi-objective optimization scheduling method based on the ant colony algorithm in cloud computing. IEEE Access **3**, 2687–2699 (2015)
20. HPE Helion Eucalyptus. https://github.com/eucalyptus