Device Centric Monitoring on Mobile Devices

Luke Chircop, Christian Colombo, Gordon J. Pace

I. Introduction

Mobile devices have become an integral part of our modern society. In fact, the use of these devices has grown to such an extent that nowadays they are finding their way to virtually every household. Various companies have also started to show interest in using such devices to support sales and productivity whilst facilitating the employees' jobs. Naturally, there are various mobile devices that one can choose from, but with a market share of over 50 percent [1] Android based devices have become a popular choice.

II. The Android Operating System

The Linux based Android [2] operating system was designed from its roots to provide the same experience across a variety of devices with different hardware and capabilities. This was achieved by building the operating system to operate in a stack like design as shown in figure 1 (ignoring the shaded part).

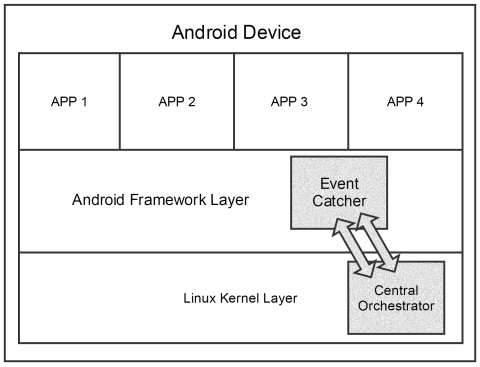

Fig. 1. Figure showing Android OS structure and Instrumentation Technique

As shown in figure 1 (ignoring the shaded part), the operating system is made up of three main layers that work on top of each other providing layers of abstraction[2]. The bottom layer consists of an adapted version of the Linux kernel which acts as an intermediate layer between the device's hardware and the software running on the device. This layer provides all the basic system functionality along with device specific drivers for hardware such as Wi-Fi and GPS. On top of this layer one finds a set of libraries and frameworks. These provide high-level services that allow applications to interface with the device specific hardware such as Wi-Fi and GPS. Finally the top layer contains all the applications that are installed on the device (example Contacts, Messaging App, etc).

The Android operating system was also designed to operate on devices with limited environment resources, therefore every application operates inside a low level (register) based machine (Dalvik Virtual Machine [3]) that acts as a blackbox. It also restricts communication between other applications and requires permission to be granted by the device owner for some features to be accessible.

III. Problem Definition

As mentioned in section II, the operating system gives device owners the ability to grant or deny any permissions that are requested by applications while they are being installed. Sometimes this is not enough since there can be cases where device owners would want to limit some of the features or access that the device user would have when using their device. For example, let us consider a parent that is concerned with how much her children spend playing games. If the parent allows the game to be installed on the device, she would not have the power to control how much time is spent playing it.

To provide such limitation and control over mobile devices Runtime Verification techniques can be used. Runtime verification [4], [5], [6] was introduced as a concept to dynamically analyze executions according to a set of predefined properties. This is achieved by weaving monitoring code inside the application or system that is going to be monitored. Therefore, every time an instrumented event executes, it is detected and the monitoring code executes allowing us to check for property violations and possibly take action.

A popular approach taken by most runtime verification tools on mobile devices is to provide users with the ability to monitor application-centric properties.

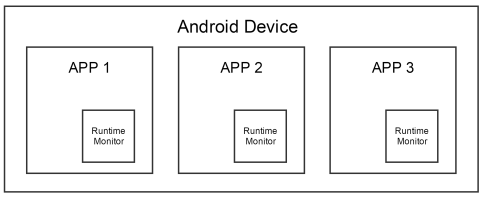

Fig. 2. Figure showing App centric instrumentation technique

This usually involves the instrumentation of runtime monitors inside applications (as seen in figure 2) while they are being installed on the device. Therefore once instrumented, the runtime monitors are able to capture events and report or

take action on any property violations. An example of some tools that are capable of instrumenting such runtime monitors include Weave Droid [1], RV-Droid [7] and Aurasium [8]. Unfortunately this approach has its share of limitations. One of which is the fact that applications are usually downloaded from the Android market and then instrumented. Therefore users can easily uninstall the instrumented application and install a clean version to bypass all the checks. Furthermore, there are a group of device-centric properties that cannot be monitored using these tools. An example of such a property is the following. Let us consider a parent that does not want her children to send more then 100 messages per hour. This property cannot be implemented using such a technique since the children could cheat by installing a number of applications capable of sending messages, hence allowing them to send more than 100 messages without triggering a property violation.

IV. Proposed Solution

Our research tackles this problem and proposes a tool that allows for device-centric properties to be monitored correctly. Amongst many properties, the proposed tool will allow device owners such as parents to for example specify how many hours their children can spend playing games. It will also be possible for device owners to specify properties that monitor browsing behavior and block potentially harmful URL requests. This technique could also be used by corporate companies to limit what applications its employees can install or use and impose files access restrictions to prevent unwanted exposure of sensitive data.

A different approach needed to be taken to achieve the goal of being able to monitor device-centric properties. Therefore, instead of instrumenting every application while it is being installed, our poposed technique instruments the Java and Android framework layer as seen in figure 1 (including the shaded part). This approach was taken since in most cases when device owners want to monitor device-centric properties, they are only concerned with interactions that occur between the application and framework layer. Using this technique also eliminated the need to instrument every application that is installed on the device.

Instrumenting the frameworks layer to catch application interactions is not enough since there can be cases where it would not be able possible to communicate or have access to any related behavior or information from other applications. This is essential when wanting to for example specify device-centric properties that do not allow for more than 100 messages to be sent per hour from the mobile phone. To tackle this problem the proposed tool also introduces a centralized orchestrator which is injected inside the kernel layer as seen in figure 1(including the shaded part) and can be reached from the frameworks layer without any permission requirements. Therefore once the events are captured, the central orchestrator is notified allowing it to determine if the event violates any predefined device-centric property and take action if necessary.

V. Conclusion

In this abstract we have identified the desire for device owners to be able to limit functionality and access to device users. It was also observed that the Android operating system did not offer such features but runtime verification techniques could be used. Existing techniques and tools were discussed and a common limitation was found. This limitation did not allow for device-centric properties to be observed. Therefore,we successfully introduced a technique that allows device owners to monitor such properties by instrumenting the framework layer and introducing a central orchestrator.

References

[1] Y. Falcone and S. Currea, "Weave droid: aspectoriented programming on android devices: fully embedded or in the cloud." in *ASE*, M. Goedicke, T. Menzies, and M. Saeki, Eds. ACM, 2012, pp. 350-353. [Online]. Available: http://dblp.unitrier.de/db/conf/kbse/ase2012.html#FalconeC12

[2] C. Nimodia and H. Deshmukh, "Android operating system," *Software Engineering, ISSN*, pp. 2229--4007, 2012.

[3] D. Ehringer, "The dalvik virtual machine architecture," 2010.

[4] C. Colombo, G. Pace, and G. Schneider, "Dynamic eventbased runtime monitoring of realtime and contextual properties," in *Formal Methods for Industrial Critical Systems*, ser. Lecture Notes in Computer Science, D. Cofer and A. Fantechi, Eds. Springer Berlin Heidelberg, 2009, vol. 5596, pp. 135-149. [Online]. Available: http://dx.doi.org/10.1007/9783642032400_13

[5] O. Sokolsky and G. Rosu, "Introduction to the special issue on runtime verification," *Formal Methods in System Design*, vol. 41, no. 3, pp. 233-235, 2012. [Online]. Available: http:dx.doi.org/10.1007/s1070301201740

[6] C. Colombo, G. Pace, and G. Schneider, "Larva safer monitoring of realtime java programs (tool paper)," in *Software Engineering and Formal Methods, 2009 Seventh IEEE International Conference on*, Nov 2009, pp. 33-37.

[7] Y. Falcone, S. Currea, and M. Jaber, "Runtime verification and enforcement for android applications with rvdroid," in *Runtime Verification*, ser. Lecture Notes in Computer Science, S. Qadeer and S. Tasiran, Eds. Springer Berlin Heidelberg, 2013, vol. 7687, pp. 88-95. [Online]. Available: http://dx.doi.org/10.1007/9783642356322_11

[8] R. Xu, H. Saïdi, and R. Anderson, "Aurasium: Practical policy enforcement for android applications," in *Proceedings of the 21st USENIX Conference on Security Symposium*, ser. Security'12. Berkeley, CA, USA: USENIX Association, 2012, pp. 27--27. [Online]. Available: http://dl.acm.org/citation.cfm?id=2362793.2362820