ISSN (Online) 2456 -1304

International Journal of Science, Engineering and Management (IJSEM)
Vol 3, Issue 4, April 2018

Run Time Investigation of Android Application

[1] T. A. MohanaPrakash, [2] K. Sathyamoorthy, [3] M. Jijendira Prasath
[1][2] Associate Professor, Department of Computer Science and Engineering, Panimalar Institute of Technology, 391, Bangalore Trunk Road
[3] Student, Department of Computer Science and Engineering, Panimalar Institute of Technology, 391, Bangalore Trunk Road

***Abstract*- Since Modern computing systems are adopted with Android operating systems there always a Need for runtime analysis in android, to reduce the gap of suitability with the real-time environments of android. It uses an agreement-based security model to prevent malware from accessing private data and prerogatives. Android universe dominates the many solutions that bear no resemblance. Underlying operating systems requires the analysis of the software platform, with the virtual machines. This paper also presents Android, a variation of Android that aims to provide real-time capabilities to Android as a whole system and the design AppAudit, an efficient analysis framework that can deliver high detection accuracy with significantly less time and memory. This paper presents a new architecture for scheduling and managing time and accuracy.**

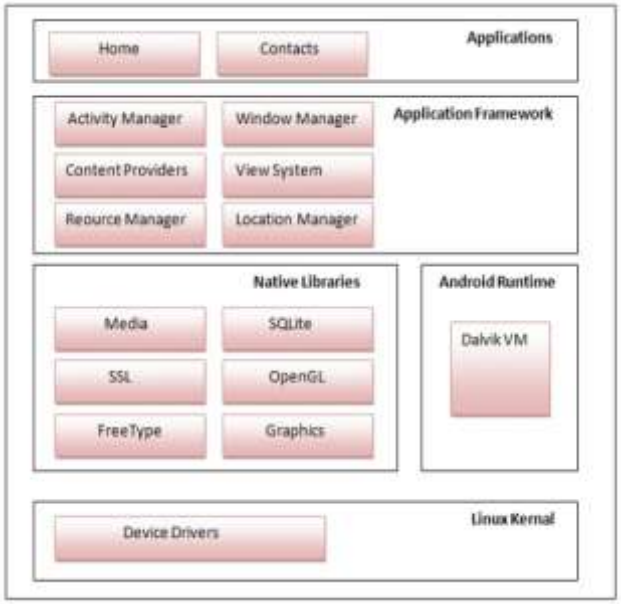
***Keywords*: Android, real-time environment, mobile applications, embedded systems.**

I. INTRODUCTION

The Android OS Oh et.al,[4] is an operating system designed for mobile platforms by Google. Android Cláudio Maia et.al, 2010[1] was made available publicly during the fall of 2008. Android is gaining strength both in the mobile industry and in other industries with different hardware architectures. The increasing interest from the industry arises from two core aspects, one is its open source nature another one is its Architectural model. Its Linux kernel-based architecture model also adds the use of Linux to the mobile industry, because of its architecture nature the knowledge and features offered by Linux are gained by android. Another important aspect is Android's own Virtual Machine (VM) environment. Android applications are Java-based. It also supports multiple real-time applications. First analyze the real-time capabilities of Android and identify limitation, then propose and implement redesigns of several internal components of Android to provide real-time support. Finally, recognize Android components, and its difficulties to evaluate every aspect of Android. Thus, the goal for this paper is to identify and redesign core components central to Android, in order to support the single real-time application. As a result of this paper discusses the potential of Android and the implementation directions to make it possible to be used in Open Real-Time environments. Wolfgang Mauerer et.al [2] said that the combined real-time Android system is able to provide remedies for both, users and programmers of embedded real-time systems.

II: ANDROID ARCHIETECTURE

Android Wolfgang et.al,[3] is an open-source software architecture. The Android platform includes an operating system, middleware and applications. Regarding the Android Runtime, besides the internal core libraries, Android provides its own VM, as previously stated, named Dalvik. Dalvik was designed from scratch and it is specifically targeted for memory-constrained and CPU constrained devices. It runs Java applications on top of it and unlike the standard Java VMs, which are stack-based, Dalvik is an infinite register-based machine. Being a register machine, it presents two advantages when compared to stack-based machines.



ISSN (Online) 2456 -1304

International Journal of Science, Engineering and Management (IJSEM)
Vol 3, Issue 4, April 2018

Figure 1. Android Architecture Namely, it requires 30% less computation time to perform instruction, which is also derived from the elimination of common expressions from the instructions. Nevertheless, Dalvik presents 35% more bytes in the instruction stream than a typical stack machine. Dalvik uses its own byte-code format name Dalvik Executable (.dex), with the ability to include multiple classes in a single file. The bottommost layer and is also a Hardware abstraction layer that enables the interaction of the upper layers with the hardware layer via device drivers. Furthermore, it also provides the most fundamental system services such as security, memory management, process management and network stack.

III: METHODILOGIES

In sections contains láudio Maia et.al, 2010[1] four possible directions to incorporate the desired real-time behavior into the android architecture. The first approach considers the system replacement of the Linux operating by one that provides real-time features and, at the same time, it considers the inclusion of a real-time VM. The second approach respects the Android standard architecture by proposing the extension of Dalvik as well as the substitution of the standard operating system by a real-time Linux-based operating system. The third approach simply replaces the Linux operating system for a Linux real-time version and real-time applications use the kernel directly. Finally, the fourth approach proposes the addition of a real-time hypervisor that supports the parallel execution of the Android platform in one partition while the other partition is dedicated to the real-time applications. Regarding the first approach, depicted in Figure 4, this approach replaces the standard Linux kernel with a real-time operating system. This modification introduces predictability and determinism in the Android architecture. Therefore, it is possible to introduce new dynamic real-time scheduling policies through the use of scheduling classes; predict priority inversion and to have better resource management strategies. However, this modification entails that all the device drivers supported natively need to be implemented in the operating system with predictability in mind. This task can be painful, especially during the integration phase. Nevertheless, this approach also leaves space for the implementation of the required real-time features in the Linux kernel. Implementing the features in the standard Linux kernel requires time, but it has the advantage of providing a more seamless integration with the remaining components belonging to the architectures involved. The second modification proposed, within the first approach, is the inclusion of a real- time Java VM. This modification is considered advantageous as, with it, it is possible to have bounded memory management. Real-time scheduling within the VM, depending on the adopted solution for better synchronization mechanisms and finally to avoid priority inversion. These improvements are considered the most influential in achieving the intended deterministic behavior at the VM level. It is important to note that the real-time VM interacts directly with the operating system's kernel for features such as task scheduling or bounded memory management. Advantages: Most of the operations provided by real-time Java VMs are limited to the integration between the VM's supported features and the supported operating system's features. Other advantage from this approach is that it is not necessary to keep up with the release cycles of Android, although some integration issues may arise between the VM and the kernel. Disadvantages: The impact of introducing a new VM is related to the fact that all the Android specificities must be implemented as well as decks support in the interpreter. Besides this is advantage, other challenges may pose such as the integration between both VMs. This integration possibly entails the formulation of new algorithms to optimize scheduling and memory management in order to be possible to have an optimal integrated system as a whole and also to treat real-time applications in the correct manner. The second proposed approach, presented in Figure 5, also introduces modifications in the architecture both in the operating system and virtual machine environments. As for the operating system layer, the advantages and disadvantages presented in the first approach are considered equal, as the principle behind it is the same. The major difference lies on the extension of Dalvik with real-time capabilities based on the Real-Time Specification for Java (RTSJ). By extending Dalvik with RTSJ features we are referring to the addition of the following API classes: RealTimeThread, NoHeapRealTimeThread, as well as the implementation of generic objects related to real-time scheduling and memory management such as Scheduler and Memory Areas. All of these objects will enable the implementation of real-time garbage collection algorithms, synchronization algorithms and finally, asynchronous event handling algorithms. However, its implementation only depends on the extent one wishes to have, meaning that a full compliant implementation may be achieved if the necessary implementation effort is applied in the VM extensions and the operating system's supported features. This extension is beneficial for the system as with it, it is possible to incorporate a more deterministic behavior at the VM level without the need of concerning about the particularities of Dalvik. Disadvantage: Having to keep up with the release cycles of the Android, especially the VM itself, if one wants Figure 5. Android Extended to add these extensions to all the available versions of the platform. Two examples of this direction are the work in states that the

ISSN (Online) 2456 -1304

International Journal of Science, Engineering and Management (IJSEM)
Vol 3, Issue 4, April 2018

implementation of a resource management framework is possible in the Android
Platform with some modifications in the platform. Although the results presented in this work are based on the CFS scheduler, work is being done to update the scheduler to a slightly modified version of EDF that incorporates reservation based scheduling algorithms. The third proposed approach, depicted in Figure 6, is also based in Linux real-time. This approach takes advantage of the native environment, where it is possible to deploy real-time applications directly over the operating system. This can be advantageous for applications that do not need the VM environment, which means that a minimal effort will be needed for integration, while having the same intended behavior. On the other hand, applications that need a VM environment will not benefit from the real-time capabilities of the underlying operating system. Finally, the fourth approach, employs a real-time hypervisor that is capable of running Android as a guest operating system in one of the partitions and real-time applications in another partition, in a parallel manner. This approach is similar to the approach taken by the majority of the current real-time Linux solutions, such as RTLinux or RTAI. These systems are able to run real-time applications in parallel to the Linux kernel, where the real-time tasks have higher priority than the Linux kernel tasks, which means that hard real-time can be used. On the other hand, the Linux partition tasks are scheduled using the spare time remaining from the CPU allocation. Disadvantage: The main drawback from this approach is that real-time applications are limited to the features offered by the real-time hypervisor, meaning that they cannot use Dalvik or even most of the Linux services. Other limitation known lies on the fact that if a real-time application hangs, all the system may also hang.

IV: RTDROID ARCHIETECTURE

In Yin et.al,[4] order to provide real-time support in all three layers depicted in, we advocate a clean-slate redesign of Android in Figure.2 Our redesign starts from the ground up, leveraging an established RTOS (e.g., RT Linux or RTEMS) and an RT JVM (e.g., Fiji VM). Upon this foundation we build Android compatibility. In other words, our design provides a faithful illusion to an existing Android application running on our platform that it is executing on Android.

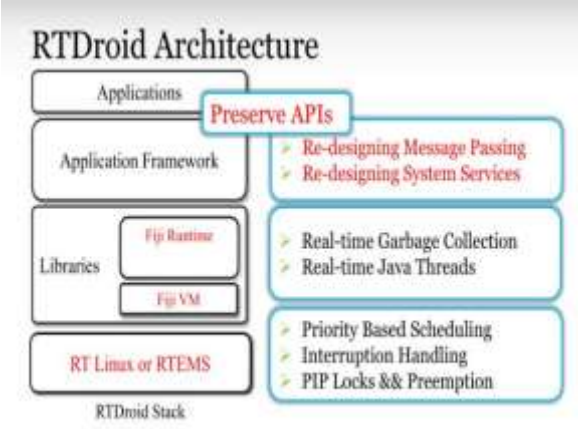


Figure 2. RTDroid Architecture This entails providing the same set of Android APIs as well as preserving their semantics for both regular Android applications and real-time applications. For real-time applications, Android compatibility means that developers can use standard Android APIs in addition to a small number of additional APIs our platform provides to support real-time features. These additional APIs provide limited Real-Time Specification for Java (RTSJ) Claudio Maia et.al,2010[1] support without scoped memory. This goal of providing Android compatibility makes our architecture unique and different from potential architecture.

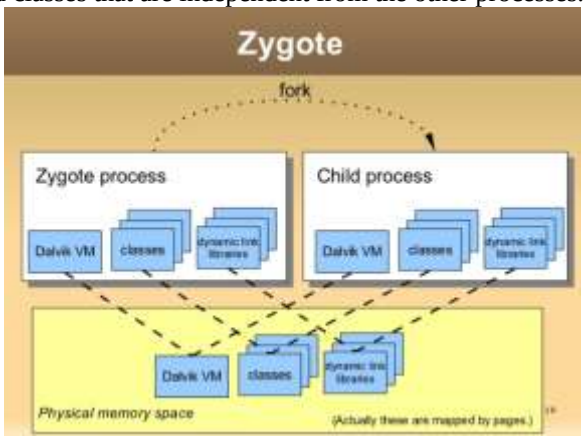
V: SUSTAINABILITY

This section discusses the suitability of Android for open embedded real-time systems, Cláudio Maia et.al, 2010[1] analyses its architecture internals and points out its current limitations. Android was evaluated considering the following topics: Its VM environment, the underlying Linux kernel, Its resource management capabilities. Dalvik VM is capable of running multiple independent processes, each one with a separate address space and memory. Therefore, each Android application is mapped to a Linux process and able to use an inter-process communication mechanism, based on Open-Binder, to communicate with other processes in the system. The ability of separating each process is provided by Android's architectural model. During the device's boot time, there is a process responsible for starting up the Android's runtime, which implies the startup of the VM itself. Inherent to this step, there is a VM process, the Zygote, responsible for the pre-initialization and pre-loading of the common Android's classes that will be used by most of the applications. Afterwards, the Zygote opens a socket that accepts commands from the application framework whenever a new Android application is started. This will cause the Zygote to be forked and create a child process

ISSN (Online) 2456 -1304

International Journal of Science, Engineering and Management (IJSEM)
Vol 3, Issue 4, April 2018

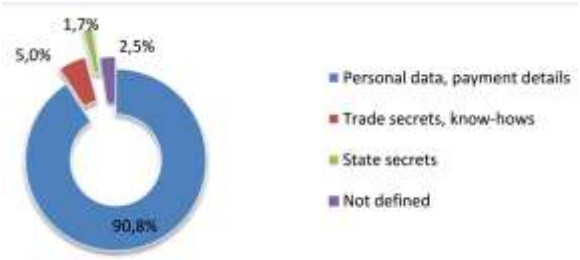
which will then become the target application. Zygote has its own heap and a set of libraries that are shared among all processes, whereas each process has its own set of libraries and classes that are independent from the other processes.



This model is presented in Figure 2. The approach is beneficial for the system as, with it, it is possible to save RAM and to speed up each application startup process. Figure 8. Zygote Heap Android applications provide the common synchronization mechanisms known to the Java community. Technically speaking, each VM instance has at least one main thread and may have several other threads running concurrently. The threads belonging to the same VM instance may interact and synchronize with each other by the means of shared objects and monitors. The API also allows the use of synchronized methods and the creation of thread groups in order to ease the manipulation of several thread operations. It is also possible to assign priorities to each thread. When a programmer modifies the priority of a thread, with only 10 priority levels being allowed, the VM maps each of the values to Linux nice values, where lower values indicate a higher priority. Dalvik follows the threads model where all the threads are treated as native threads. Internal VM threads belong to one thread group and all other application threads belong to another group. According to source code analysis, Android does not provide any mechanisms to prevent priority inversion neither allow threads to use Linux's real-time priorities within Dalvik.

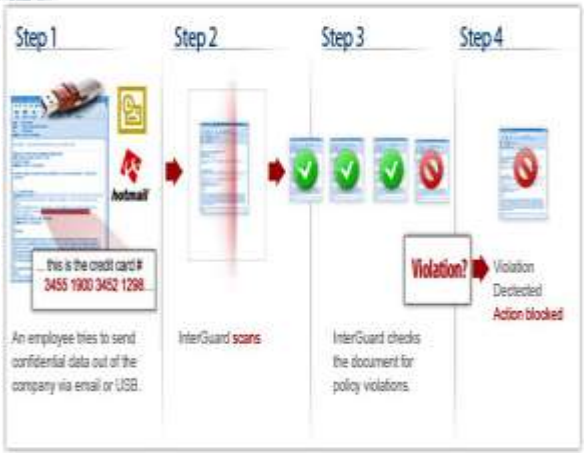
VI: FINDING OF REAL-TIME ANDROID

5.1 Finding 1: Mingyuan et.al, [5] Most data leaks are caused by 3rd-party advertising libraries: From Table 1.1, we found that 28 out of the 30 (93.3%) detected data leaks are caused by 3rdparty advertising libraries. As previous research [9], has pointed out, 3rd-party advertising Modules aggressively request application permissions to access various personal data. If an advertising library leaks data, it can potential affect lots of apps. Meanwhile, hackers have started to exploit advertising libraries to spy on users. We believe that privilege separation and fine-grained privilege control will help to prevent the threats caused by these problematic libraries. From the perspective of app developers, App Audit can help check their apps before publishing to the market, which could effectively detect data leaks beforehand and avoid accidentally using data-leaking 3rd-party modules.



VII: COMPARISON REAL-TIME APPS:

Free apps that spread certain personal information identified by AppAudit. In the following table, for the "Privacy Policy" column, a "lib" means that the privacy policy does not cover the kind of data spread by advertising libraries. From Table1.1, we found that 28 out of the 30 (93.3%) detected data leaks are caused by 3rdparty advertising libraries. As previous research [5], has pointed out, 3rd-party advertising modules aggressively request application permissions to access various personal data. If an advertising library leaks data, it can potential affect lots of apps.



Meanwhile, Zhang et.al,[7] said that hackers have started to exploit advertising libraries to spy on users. We believe that privilege separation and fine- grained privilege control will help to prevent the threats caused by these problematic libraries. From the perspective of app developers, AppAudit can help check their apps before publishing to the market,

which could effectively detect data leaks beforehand and avoid accidentally using data-leaking 3rd-party modules. From the table, we can find that, apps (Word Search and Speed test) are gaining awareness of privacy by removing problematic advertising libraries. We believe that AppAudit, when integrated with IDEs, could well assist developers for this purpose. On the other hand, we discover advertising libraries are gaining privacy awareness as well. For example, a newer version of the Tap joy advertising library hashes. IMEI before sending it to the advertising server.

IX. CONCLUSION

Android OS supports pre-emption and multi-tasking, the results indicate Android may be seen as a potential target for real-time environments and there are numerous industry targets that would benefit from architecture with such capabilities. Taking this into consideration, this paper presented the suitability of the Android as a real-time system. By focusing on the core parts of the system it was possible to expose the limitations and to present four possible directions that add real-time behavior to the system. This paper also presented RTDroid, a variation of Android that aims to provide real-time capabilities to Android as a whole system. We have shown that replacing DVM with an RT JVM and Linux with an RTOS is insufficient to run an Android application with real-time guarantees. In this paper, the design AppAudit, an efficient analysis framework that can deliver high detection accuracy with significantly less time and memory. AppAudit comprises a static API analysis that can effectively narrow down analysis scope and an innovative dynamic analysis which could efficiently execute application byte code to prune false positive and confirm data leaks. To address this shortcoming, we have redesigned Android's core constructs and system services to provide tight latency bounds to real-time applications to be useful for the that propose to use Android OS.

REFERENCE

[1] Cl ´ audio Maia, Lu´ ıs Nogueira, and Luis Miguel Pinho, "Evaluating Android OS for embedded real-time systems", pages 63–70, 2010

[2] Wolfgang Mauerer, Gernot Hillier, Jan Sawallisch, Stefan H onick and Simon Oberth,"Real-Time Android: Deterministic Ease of Use" University of Paderborn, Germany.

[3] Yin Yan, Shaun Cosgrove, Varun Anand, Amit Kulkarni, Sree Harsha Konduri, Steven Y. Ko, Lukasz Ziarek, "Real-Time Android with RTDroid",University of New York, avairable @buffalo.edu

[4] Hyeong-Seok Oh, Beom-Jun Kim, Hyung-Kyu Choi, and Soo-Mook Moon, "Evaluation of Android Dalvik virtual Machine", pages 115–124, 2012.

[5] Toward Making jPapaBench Fly: An Experience Report Shaun Cosgrove, Yin Yan, Sai Tummala, Manish Jain, Karthik Dantu, Steven Y. Ko, Lukasz Ziarek

[6] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment,"J. ACM, vol. 20, no. 1, pp. 46–61, 1973.

[7] X.Zhang,A. Ahlawat, and w. Du, "Isolating advertisements from mobile applications in android", pages 9-18, 2013.

[8] Coderre, D. G., Computer-Aided Fraud Prevention and detection: A Step- by-Step Approach , John Wiley and Sons, 2009.