Poster: FireDroid: Hardening Security in Android with System Call Interposition

Giovanni Russello, Arturo Blas Jimenez, Habib Naderi, Wannes van der Mark
Department of Computer Science
University of Auckland
Auckland, New Zealand
Faculty Member
Email: {g.russello;a.blas;h.naderi;w.vandermark}@auckland.ac.nz

***Keywords*-Android security; system call interposition;**

I. Introduction

Smartphones are the most successful consumer devices to date reaching 821 million units sold to end users in the last quarter of 2012 [1]. In this very competitive market, smartphones equipped with the Android OS reached the 72.4% of the total sold devices [2]. According to Google chairman Erich Schmidt, 1.3 million Android devices are activated each day [3]. As these simple statistics testify, the Android OS plays an important role in the mobile device market.

Smartphones empower users with ubiquitous computing: users can perform the same tasks as with laptops and desktops but with more flexibility and mobility. Both in the private and public sector, organisations' employees rely on their smartphones for managing professional and personal computation by installing business and entertainment applications. As a consequence, these users require that their personal smartphones are connected to their work IT infrastructure. Organisations are willing to support employee-owned smartphones because of the increase in productivity of their employees. Bring-Your-Own-device policy is getting very popular within the business sector. Android penetration in the business sector is on the rise and poses serious security challenges to IT departments and CIOs [1].

However, because users can install third-party applications on their smartphones, several security concerns may arise. For instance, malicious applications can upload to remote servers private information stored on the device such access contact list, SMS and MMS. By exploiting pre-installed application vulnerabilities [4], malware can reset the device to its factory settings deleting all the data that it contains. This poses serious security concerns to sensitive corporate data, especially when the standard security mechanisms offered by the platform are not sufficient to protect the users from such attacks. Alternative solutions such as anti-virus software are not able to cope with the rapid evolution of malware. As reported by Zhou and Jiang in [5] the best anti-virus software still was not able to detected 21% of the malware currently in the wild.

In this paper, we propose FireDroid an effective security solution for enforcing fine-grained security policies without the need to recompile any internal modules of the Android OS. FireDroid monitors the execution of system calls and it is able to effectively confine the execution of processes within a secure sandbox. FireDroid exploits the unique mechanism in Android for spawning applications in order to automatically monitor any applications executed in an Android device. In this way, FireDroid does not require to modify the code of an application for being able to enforce security policies. As a main advantage compared to other approaches, FireDroid is able to monitor any application and system code executed in a device. This makes FireDroid very effective also for controlling the execution of native code. We have implemented FireDroid and evaluated its effectiveness against a large set of real malware samples. Moreover, we show FireDroid can be used to enforce policies to stop attacks that exploit vulnerabilities of pre-installed applications and Android system services.

II. FireDroid System Design

The main observation behind our approach is that although most of the Android applications do not interact directly with the Linux kernel, all their privileged operations rely on system calls executed by the Linux kernel. As such, by controlling the execution of system calls it is possible to control the behaviour of applications.

FireDroid provides an interposition mechanism for interposition of system calls generated by the applications and enforcing security policies to control their execution. FireDroid can be thought as a network firewall controlling the interactions between applications and kernel through system calls.

There are several ways in which system call interposition can be realised. Because our main goal is to extend the Android security without modifying applications, the Android middleware, and underlying Linux OS, we decided to use the `ptrace()` system call for tracing applications' executions. When FireDroid is deployed, each Linux process is monitored by a **FireDroid Application Monitor (FDAM)**. As shown in Figure 1, the FDAM attaches to the target process through `ptrace()` system call meaning that each time the target process executes a system call, the kernel suspends the target process and notifies the FDAM. Within the FDAM, the **Policy Enforcement Point (PEP)** is responsible for gathering the required information from the system call executed by the target process including the parameters in the system call.

The PEP forwards this information to the **Policy Decision Point (PDP)** that will retrieve the relevant policies from the **Policy Repository (PR)** within the FDAM. The PR contains policies specific to the process being monitored. Depending on the policies, the PDP can decide to either allow or deny the execution of the system call. Moreover, the policy evaluation might also return to kill the target process. Another possible outcome of the policy evaluation is to inform the user and to ask his decision (either allow, deny or kill the process). To ask the user, the PDP contacts the **User Notification Component** in the **FireDroid Service (FDS)**. The FDS also provides a **Policy Administration Point (PAP)** to manage security policies defined at device level. In FireDroid, the user can dis/enable policies that can be applied to her device. However, in a BYOD scenario, enterprise-specific policies can not be managed by the user. In this case, the security administrator can remotely manage the security policies by sending updated policies. The **Remote Policy Manager (RPM)** component handles the remote edit/update requests of policies. New policies can be sent through SMS/MMS and/or Bluetooth and stored in the **Global Policy Repository**. The PAP is responsible for pushing policy updates to the PRs of each FDAM.

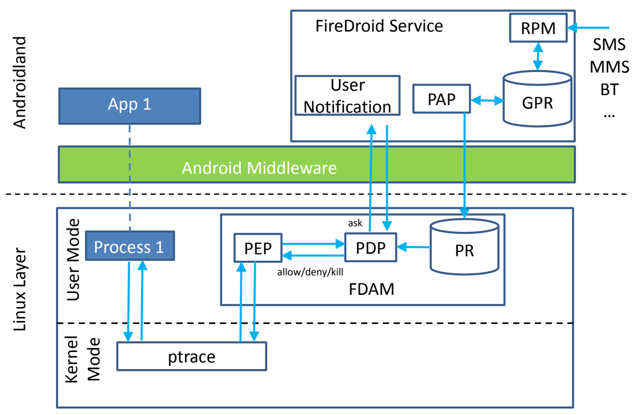

Fig. 1. FireDroid Details.

III. Implementation Details

FireDroid controls applications' behaviour by monitoring their interaction with the OS. Other approaches, such as Janus [6] and Systrace [7] have been proposed for system call interposition for the Linux OS in fully-fledged machines. The main difference between FireDroid and both Systrace and Janus is related to the unique mechanism Android has for starting applications. This mechanism makes the attachment of Systrace and Janus to applications impractical in Android.

In more details, Systrace and Janos rely on the user to use a shell to attach their monitoring process to a target application. This attaching mechanism is not possible in Android where applications are not launched as in typical Linux box. Android applications are launched through the **Zygote** process. Zygote is the only process authorised to fork a new process. Zygote is started when Android is booted and represents the initiator of all the applications that will be launched. Each time a new application is started, Zygote forks a new process copying its initialised structures. The sharing of this already linked libraries as well as the initialized DVM and all the components needed by an Android component to run allows a faster start-up of the new application.

In FireDroid, we exploit this mechanism for automatically attach to any applications launched in Android. We have implemented a monitor process called **FireDroid Main Monitor (FDMM)**. The FDMM is only responsible for monitoring when Zygote is executing a `fork()` system call. To be able to monitor Zygote, the FDMM has to be its parent process. Because Zygote is started when Android is booted, we have to attach the FDMM to Zygote at booting time modifying the Android starting up sequence. This is the only modification required to enable FireDroid in an Android device. No modification and recompilation of the Android Open Source Project (AOSP) is require to have FireDroid fully functional in Android.

IV. Demo

We have fully implemented FireDroid and deployed it in several devices including the Samsung Galaxy S3, Samsung Galaxy Note II and HTC One X. During the Poster session at the conference, we will provide a demo of FireDroid functionality demonstrating no noticeable performance degradation when FireDroid is enabled on a device.

V. Acknowledgements

We would like to thank Xiao Clark, Daniel Lewis, and Kris Pritchard for their contributions to the realisation of FireDroid and its demo. We also would like to extend our thanks to Auckland UniServices Limited for its financial support.

References

[1] http://www.gartner.com/it/page.jsp?id=2227215.
[2] http://www.gartner.com/it/page.jsp?id=2237315.
[3] http://news.cnet.com/8301-1035_3-57545513-94/five-years-of-android-by-the-numbers/.
[4] http://www.computerworld.com/s/article/9231758/USSD_attack_hit_SIM_cards_and_Samsung_Android_devices.
[5] Y. Zhou and X. Jiang, "Dissecting android malware: Characterization and evolution," in *IEEE Symposium on Security and Privacy*, 2012, pp. 95–109.
[6] D. A. Wagner, "Janus: an approach for confinement of untrusted applications," EECS Department, University of California, Tech. Rep. UCB/CSD-99-1056, 1999.
[7] N. Provos, "Improving host security with system call policies," in *Proc. of the 12th conf. on USENIX Security Symposium*, vol. 12, Washington, DC, 2003, pp. 18–18.