Available online at www.sciencedirect.com

SciVerse ScienceDirect

journal homepage: www.elsevier.com/locate/cose

Computers & Security

Breaking and fixing the Android Launching Flow

Alessandro Armando [a,b], Alessio Merlo [a,c,*], Mauro Migliardi [d], Luca Verderame [a]

[a] DIBRIS, Università degli Studi di Genova, Italy
[b] Security & Trust Unit, FBK-irst, Trento, Italy
[c] Università e-Campus, Italy
[d] DEI, University of Padova, Italy

ARTICLE INFO

Article history:
Received 24 November 2012
Received in revised form
13 March 2013
Accepted 15 March 2013

Keywords:
Android OS
Android security
Android security framework
Zygote vulnerability
Denial-of-Service

ABSTRACT

The security model of the Android OS is based on the effective combination of a number of well-known security mechanisms (e.g. statically defined permissions for applications, the isolation offered by the Dalvik Virtual Machine, and the well-known Linux discretionary access control model). Although each security mechanism has been extensively tested and proved to be effective in isolation, their combination may suffer from unexpected security flaws. We show that this is actually the case by presenting a severe vulnerability in Android related to the application launching flow. This vulnerability is based on a security flaw affecting a kernel-level socket (namely, the Zygote socket). We also present an exploit of the vulnerability that allows a malicious application to mount a severe Denial-of-Service attack that makes the Android devices become totally unresponsive. Besides explaining the vulnerability (which affects all versions of Android up to version 4.0.3) we propose two fixes. One of the two fixes has been adopted in the official release of Android, starting with version 4.1. We empirically assess the impact of the vulnerability as well as the efficacy of the countermeasures on the end user. We conclude by extending our security analysis to the whole set of sockets, showing that other sockets do not suffer from the same vulnerability as the Zygote one.

1. Introduction

By leveraging a generic Linux kernel, the Android OS is built out of a layered architecture that runs on a wide variety of devices and supports the execution of a large number of applications available for download both inside and outside the Google Play Store. Since most applications are developed by third-parties, security is a major concern. The Android security model tackles the problem by striving to attain the following design goal:

> *A central design point of the Android security architecture is that no application, by default, has permission to perform any operation that would adversely impact other applications, the operating system, or the user.*http://developer.android.com/guide/topics/security/security.html

This goal is pursued through a number of cross-layer security mechanisms aimed at isolating applications from each other. These mechanisms are built out of basic security mechanisms available in the individual layers of the Android

* *Corresponding author*. DIBRIS, Università degli Studi di Genova, Via all'Opera Pia, 13, 16145 Genova, Italy. Tel.: +39 (0)10 3532344.
E-mail addresses: alessandro.armando@unige.it, armando@fbk.eu (A. Armando), alessio.merlo@uniecampus.it, alessio.merlo@unige.it (A. Merlo), mauro.migliardi@unipd.it (M. Migliardi), luca.verderame@unige.it (L. Verderame).
0167-4048/$ – see front matter © 2013 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.cose.2013.03.009

stack. For instance, *application sandboxing*[1] is achieved by combining the isolation guaranteed by the use of Dalvik Virtual Machines together with the native Discretionary Access Control (DAC) offered by Linux and with a set of statically defined *Android permissions* specifying which operations each application is allowed to execute. These permissions regulate access to hardware components such as the telephony subsystem as well as accesses to sensitive data (e.g. messaging history) and to inter-process communication mechanisms (e.g. intent and broadcast). Each Android application comes up with a set of permissions (i.e. corresponding to operations the application requires to execute during its life cycle) that the user must explicitly grant during installation or upgrade. According to their potential risk, Android permissions are divided into four categories, namely (from lower to higher risk) *normal*, *dangerous*, *signature* and *signatureOrSystem*.[2]

The individual mechanisms are generally well-known and have been thoroughly tested i.e. the isolation offered by virtual machines has been tested for Java applets, access control for Unix/Linux is well-known, and explicit permissions have been tested inside the Java security architecture. However, this is not the case for the specific combination deployed in Android: the cross-layer interaction among the available mechanisms has not been fully explored yet and may therefore suffer from security weaknesses.

In this paper we present a serious vulnerability in the Android Launching Flow. The *Android Launching Flow* is a sequence of activities that are normally carried out by the system whenever a new application is launched. In Android, applications cannot directly fork new processes and a single process, the *Zygote Process*, is permitted to carry out this crucial activity. Requests for the creation of new processes are therefore sent to the *Zygote Process* through a specific socket, called the *Zygote Socket*.

Up to OS version 4.0.3 included, the Android Launching Flow suffered from a vulnerability that allows a malicious application to side-step all Android security checks and force the system to fork an unbounded number of processes thereby making the device completely unresponsive. In order to exploit the vulnerability, a malicious application does not need any Android permission.

Rebooting the device does not necessarily help as a malicious application can be crafted in such a way to be automatically launched at boot-time without asking for the user's explicit approval of any permission upon installation. This makes the malicious applications particularly mischievous as even the most cautious user may consider it as harmless.

More in general, the existence of this vulnerability shows that the aforementioned fundamental design goal of the Android Security Framework, i.e. the impossibility for an application to adversely impact another application, is not met. Indeed, a malicious application can severely affect all other applications, the operating system, and ultimately the user's experience.

In the paper we also propose two solutions to the problem: the first solution involves minimal changes in the Android security model, the second requires a number of additional cross-layer checks. We show that both solutions are equally effective in fixing the vulnerability.

In the paper we also report on experimental results confirming that:

- all versions of Android OS up to 4.0.3 included (that means 80% of Android devices in March 2013) suffer from the vulnerability,
- our proposed fixes effectively counter the DoS attack, and
- the two most recent versions of Android (namely, versions 4.1 and 4.2) no longer suffer from the vulnerability.

We have promptly informed of our findings both the Android Security Team and the US-CERT. In response to our findings, the Android OS has been swiftly patched by incorporating (a simplified variant of) one of our two fixes in the two most recent versions of Android. Moreover, the US-CERT has recently issued a vulnerability note (CVE-2011-3918) that describes the issue and evaluates its severity as HIGH.[3]

This paper revises and extends our previous work (Armando et al., 2012b) in a number of ways:

- we provide a more detailed analysis of the Android architecture and of its security mechanisms;
- we provide an extended description of the vulnerability as well as of the countermeasures, by adding details that are necessary to fully understand and reproduce the problem, and the proposed fixes;
- we extend the experiments related to both the vulnerability and the countermeasures by involving end users;
- we provide a new section describing the recent developments in the Android architecture related to the discussed vulnerability and the patch adopted in the latest versions of Android;
- we assess the security of two additional sockets used by Android that are very similar to the Zygote socket.
- we report the results of an experimental analysis that confirms the effectiveness of the fixes in countering the DoS attack and that the patches do not affect the nominal behavior of the system.

1.1. Structure of the paper

In the next section we provide a brief description of the Android architecture. In Section 3 we describe the security mechanisms used in Android. In Section 4 we present the vulnerability and in Section 5 we illustrate two possible solutions. In Section 6 we present our experimental results. In Section 7 we discuss the effectiveness of the fix incorporated in Android and extend our security analysis to other parts of Android. In Section 8 we compare our work with the current Android literature and we conclude in Section 9 with some final remarks.

[1] http://developer.android.com/guide/practices/security.html.

[2] http://developer.android.com/guide/topics/manifest/permission-element.html.

[3] http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2011-3918.