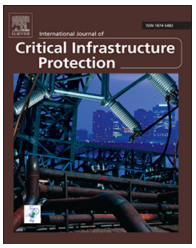
Available online at www.sciencedirect.com

ScienceDirect

www.elsevier.com/locate/ijcip



Security considerations related to the use of mobile devices in the operation of critical infrastructures

Alessandro Armando, Alessio Merlo, Luca Verderame*

Computer Security Laboratory, DIBRIS – University of Genoa, Viale F. Causa 13, 16145 Genoa, Italy

ARTICLE INFO

Article history:
Received 8 January 2014
Accepted 4 October 2014
Available online 17 October 2014

Keywords:
Mobile devices
Android
Malware
Cross-layer interplay
Security
Critical infrastructure

ABSTRACT

An increasing number of attacks by mobile malware have begun to target critical infrastructure assets. Since malware attempts to defeat the security mechanisms provided by an operating system, it is of paramount importance to understand the strengths and weaknesses of the security frameworks of mobile device operating systems such as Android. Many recently discovered vulnerabilities suggest that security issues may be hidden in the cross-layer interplay between the Android layers and the underlying Linux kernel. This paper presents an empirical security evaluation of the interactions between Android layers. The experiments indicate that the Android Security Framework does not discriminate between callers of invocations to the Linux kernel, thereby enabling Android applications to directly interact with the kernel. This paper shows how this trait allows malware to adversely affect the security of mobile devices by exploiting previously unknown vulnerabilities unveiled by analyses of the Android interplay. The impact of the resulting attacks on critical infrastructures is discussed. Finally, an enhancement to the Android Security Framework is proposed for detecting and preventing direct kernel invocations by applications, thereby dramatically reducing the impact of malware.

1. Introduction

Mobile devices are no longer used only for personal and leisure activities; they are finding their way into organizations and businesses. Since mobile operating systems are geared for personal use, their use in other environments is raising serious security concerns, especially when mobile devices are used to monitor and control critical infrastructure assets. Although a variety of organizations have become very interested in the Bring Your Own Device (BYOD) paradigm [14] and several solutions have been put forward [5,22,37], security concerns persist and the paradigm has yet to gain wide acceptance.

Mobile devices are also already routinely used by operators to monitor and control critical infrastructures. For example, in Haverhill, Massachusetts, on-call workers use tablet PCs to monitor alarms and other systems in the local water treatment plant that serves 58,000 customers [23]. But many more solutions addressing a variety of application scenarios are emerging, including a collaborative indoor positioning system designed for time-critical scenarios (e.g., military operations) where infrastructure-based localization is impossible [30] and a mobile radiation detector with the ability to log and share data using 3G and GPS technologies [25]. The transition from personal to professional use of mobile devices is driving more stringent security requirements. Needless to say, fulfilling these requirements is crucial when mobile devices are used in the critical infrastructure.

As pointed out in a 2013 Mobile Threat Report by F-Secure Labs [20], malware is the primary threat facing mobile

*Corresponding author.
E-mail address: alessio.merlo@unige.it (A. Merlo).

http://dx.doi.org/10.1016/j.ijcip.2014.10.002
1874-5482/

devices. Malicious apps can be retrieved from application markets (e.g., Apple Store, Google Play, Samsung Store and Windows Store) and installed on mobile devices. This malware exploits vulnerabilities in other applications or in the mobile operating system itself. When executed, it may steal sensitive information, corrupt the integrity of data, and even affect device usability. Recent studies [29] have shown that it is not particularly difficult for a programmer to implement and distribute malware using a public store. The deployment of countermeasures at the market-side (see, e.g., [3,27]) may mitigate adverse effects, but it does not eradicate malicious applications [31,38].

This paper focuses on the highly popular Android operating system. Android engages a Java stack built on top of a native Linux kernel. The services and their functionality are realized through the interplay of components residing in different layers of the operating system that invoke function calls. The Android Security Framework (ASF) stems from the fruitful combination of security mechanisms in different layers, namely the standard discretionary access control (DAC) model of Linux, the isolation offered by the Java Virtual Machine (JVM), and a collection of Android-specific mechanisms such as the Android permission system. The Android Security Framework oversees the cross-layer interplay among components to detect malicious or unwanted interactions and intervene, if necessary. However, the security that it offers has been recently challenged by the discovery of a number of vulnerabilities involving different layers of the Android stack and the associated cross-layer interplays (see, e.g., [6,12,16]).

One example is the Zygote vulnerability reported in [6], which enables malware to force a Linux kernel to fork an unbounded number of processes, rendering the device completely unresponsive. In this case, the problem arises from the fact that the Android Security Framework is unable to discriminate between a legitimate Android cross-layer interplay performed by trusted Android services and an insecure interplay involving an application; this permits the direct invocation of a critical kernel functionality (fork operation) by any application. The essence of the problem is the lack of control on Linux system calls that are involved in launching new applications.

This paper examines if the lack of control between the Android stack and Linux kernel is limited to certain types of calls or if it is a more general issue with the Android Security Framework. If the latter is true, there is the distinct fear that malware could leverage a wide attack surface that spans the entire native Linux kernel. To ascertain the situation, an empirical assessment is conducted for the cross-layer interplay between the Android stack and Linux kernel. The assessment provides insights into the extent to which the Android Security Framework can discriminate between trusted and untrusted invocations of core system functionality. The results demonstrate that very little control is exercised by the Android Security Framework and that malware may force and exploit insecure interplays, including attacks that adversely affect privacy and device usability. This paper also discusses the impact that such Android-based malware may have on mobile devices that are used in the critical infrastructure. Finally, an enhancement is proposed to the Android Security Framework that helps recognize and effectively prevent insecure cross-layer interplays between the Android stack and Linux kernel.

2. Related work

Android security has received considerable attention during the past few years. For example, general surveys are available in [18,35], vulnerabilities are reported in [6,33] and vulnerability detection techniques are proposed in [12,21]. Most of the research focuses on enhancing the Android architecture and its security model.

Researchers have shown that the Android platform suffers from vulnerabilities that allow malware to perform denial-of-service attacks [6], create covert channels [33] and launch web attacks [28] and privilege escalation attacks [16]. Some vulnerabilities are intrinsic to the Android security model (see, e.g., [16]), while others are due to security flaws, bugs and the lack of controls in the security mechanisms that constitute the Android Security Framework (see, e.g., [6,28,33]).

Several tools have been developed for malware detection and application certification. For example, SCanDroid [21] performs automatic application certification by assessing whether application data flows are consistent with the application manifest. Comdroid [12] assesses the actual privileges of Android applications by analyzing their intent-based communications. Another tool [34] statically analyzes Android executables to compare function calls with malware signatures.

Solutions for malware detection at the market-side have also been developed. An example is Google Bouncer [27], a malware detector that runs on the Google Play store; however, Oberheide and Miller [31] have shown that Bouncer is easily circumvented. DroidRanger [40] is an application market analyzer that combines a footprint-based detection engine of known malware, focusing on API calls and declared permissions, with a heuristic detection engine for zero-day malware. These tools are powerful, but they do not consider cross-layer interplays between the Android stack and Linux kernel or direct kernel invocations via system calls, which is the focus of this paper.

The monitoring of Android system calls is performed by the Android Application Sandbox (AASandbox) [8] and by Crowdroid [10]. AASandbox can perform static and dynamic analyses in a fully isolated environment through a loadable kernel module that monitors system calls. Crowdroid carries out a dynamic analysis of application behavior. Based on the collaboration with the Android user community, Crowdroid can distinguish between benign and malicious applications of the same name and version, detecting anomalous patterns in the execution of system calls. These approaches are different from the one presented in this paper for two reasons: (i) they log only the return value of each system call without logging the parameters and (ii) they do not provide security assessments of system calls, nor propose solutions that could mitigate possible malicious interplays.

Formal frameworks for modeling application behavior in terms of interactions have been proposed. Chaudhuri [11] has proposed a language-based approach to infer the security properties of Android applications. Although it allows for the formal assessment of application properties, the approach does