Enabling Secure and Trusted Device I/Os on Smartphones

BINGHAMTON UNIVERSITY
State University of New York

PI: Yifan Zhang

Department of Computer Science, SUNY Binghamton, Binghamton, NY

zhangy@Binghamton.edu

Supported by NSF Award # 1566375 (CRII)

Why device I/O matter ?

Three common categories of attacks through built-in smartphone I/O devices (e.g., touchscreens, sensors, cameras, and microphones):

- Attackers can passively infer or actively steal user info from smartphone device inputs.
- Attackers can tamper with or forge smartphone device inputs to disrupt services relying on those input data or even gain control of the smartphone.
- Attackers can trick users to reveal their sensitive information through UI spoofing or task hijacking attacks by manipulating smartphone touchscreen device outputs.

Objectives

- Objective 1: Securing device inputs against even attackers with root privileges to prevent device inputs inference or stealing.
- Objective 2: Enabling trusted device inputs so that protected applications are guaranteed that the device inputs delivered to them have not tempered with.
- Objective 3: Providing trusted device outputs to allow users to identify the authenticity of an app's screen output.

Challenges

Our project aims to achieve the above objectives with ***practical*** and ***scalable*** solutions, which pose the following notable challenges:

- Supporting existing unmodified operating systems.
- Supporting unmodified smartphone applications.
- Achieving good system performances, as well as good user experiences.

Our approach

Our approach is IOGuard, a system to enable secure and trusted device I/Os on smartphones. IOGuard achieves the 3 objectives above with an integrated and systematic approach.

Two core components of IOGuard system:

- a small and dedicated bare-metal hypervisor built using the recently introduced ARM hardware virtualization support, and
- a user-space sandbox framework that enables running and protecting unmodified applications.

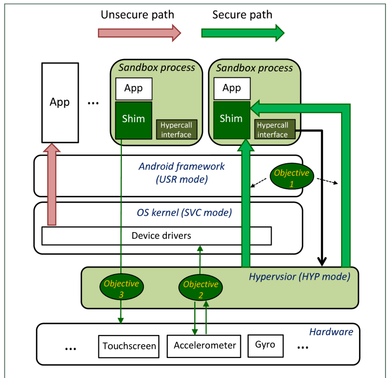


Figure 1. IOGuard overview

IOGuard overview

Figure 1 shows the architecture of IOGuard on Android. Two major components (shaded in light green):

- A dedicated bare-metal hypervisor
    - Higher privileged than OS and apps → monitor I/O operations by trapping and inspecting sensitive I/O activities (without changing the OS).
- A user-space sandbox framework
    - Unmodified apps run in a sandbox process.
    - The shim intercepts and adapts I/O communications between the app and the Android framework to allow the app to run normally.

Three tasks to fulfill the three functionality objectives.

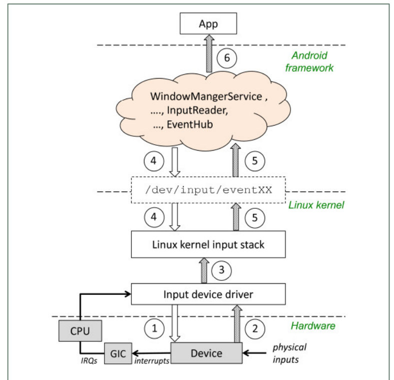


Figure 2. Background: device inputs in Android

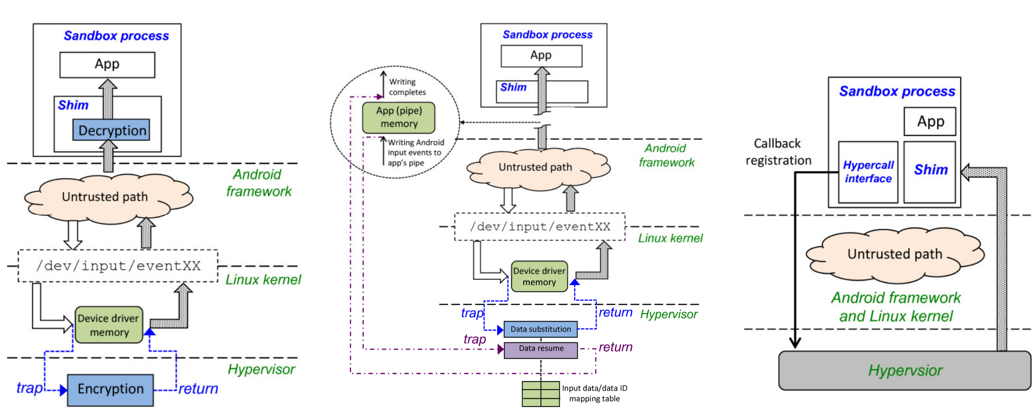
Background: device inputs in Android

Figure 2 shows the device inputs handling in Android. The Linux input subsystem is used for low level handling:

- Interrupts are used to signal physical input events.
- MMIO by INT handler to obtain the event data (①&②).
- Linux kernel input stack manages all input events (③).
- The kernel input stack exposes device input events to user space via device nodes (/dev/input/eventXX).

A complex Android input stack spanning across the layers of Android native libraries and the application framework. But in a nutshell:

- The framework polls kernel for device inputs (④).
- The framework adapts the data returned by kernel (⑤) into Android input events, and deliver them to apps through app callbacks (⑥).



(a). Encryption based endpoint communication protocol

(b). Data substitution based endpoint communication protocol

(c). Direct access based endpoint communication protocol

Figure 3. Building secure input path between input devices and applications (i.e., the endpoints).

Task 1: building secure device input path

Two different types of approaches:

- To hide the input data from the untrusted entities.
- To allow direct communication between the two trusted endpoints (i.e., the input device & the app)

Three different endpoint communication protocols:

- Encryption-based communication (Figure 3 (a)).
- Data-substitution-based communication (Figure 3 (b)).
- Direct-access-based communication (Figure 3 (c)).

Other issues:

- User-space sandboxing framework design to support unmodified apps.
- Protecting code of hypervisor and sandbox framework.
- Protecting sandbox process's memory from kernel.

Task 2: realizing trusted device data reading

Having a secure path alone is not sufficient, we need to:

- prevent I/O device misconfiguration (unintentional or intentional).
- Ensure that the device driver has faithfully placed the input data read into the device driver memory without disclosing to other untrusted parties.

Our solution:

- Hypervisor traps and inspects MMIO activities related to device configuration.
- Move the part of reading event data into kernel input stack memory from device drivers to hypervisor (through trapping and serving I/O device interrupts).
- Write-protecting kernel input stack memory to prevent data injection attack.

Task 3: enabling trusted screen output

Goal: provide trusted screen output to defeat UI spoofing and task hijacking attacks.

Our solution: allow hypervisor to monitor and control the hardware framebuffer such that trusted screen output is enforced.

Ongoing implementation and evaluation

- Implementation and evaluation platform validation: Odroid XU3 development board featuring Samsung Exynos 5422 SoC (ARMv7).
- User-space application sandboxing framework.
- Actively working on the implementation of info-hiding based secure device input path.