Evading Android Emulator

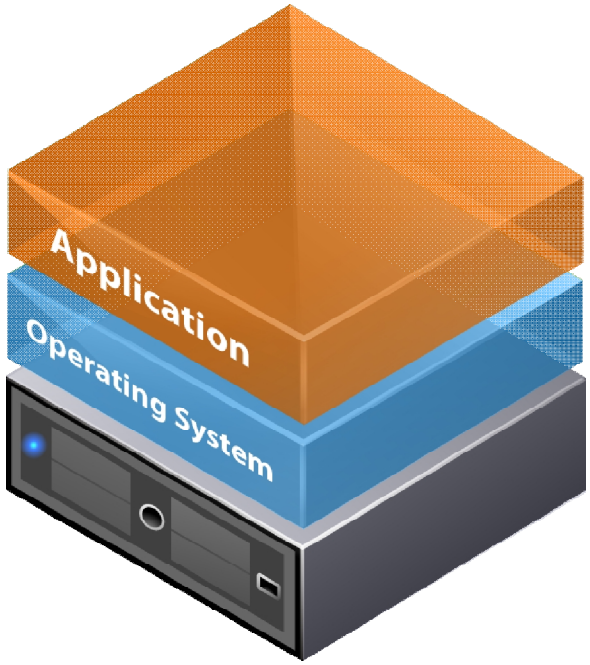
Thanasis Petsas

petsas@ics.forth.gr

What is a Virtual Machine?

- A software based computer that functions like a physical machine
- A VM typically has two components
  - **Host OS**: The host server that provides computing resources
  - **Guest OS**: a separate and independent instance of an OS
- **Hypervisor**: a layer between host and guest
  - Isolates each guest OS from another

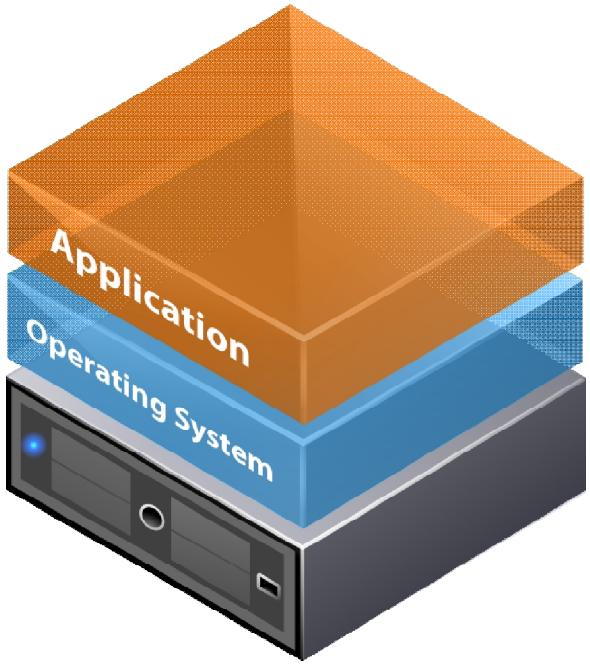
Traditional vs. Virtual Architecture



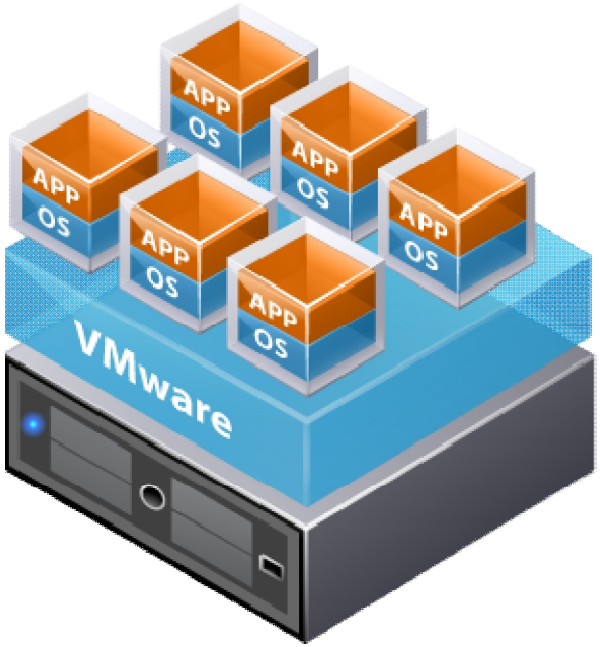
Traditional Architecture

(Quote from VMware)

Traditional vs. Virtual Architecture



Traditional Architecture



Virtual Architecture

(Quote from VMware)

Why to Use a Virtual Machine ?

- Application isolation
- Consolidation
  - Better use of the Hardware
- Reduced energy consumption
- Portability
  - Quick image cloning, running a new instance
- Manageability
- Better Software development and testing

Android Emulator

- Can run virtual mobile devices on a computer
- mimics *all* of the hardware and software features of a typical mobile device

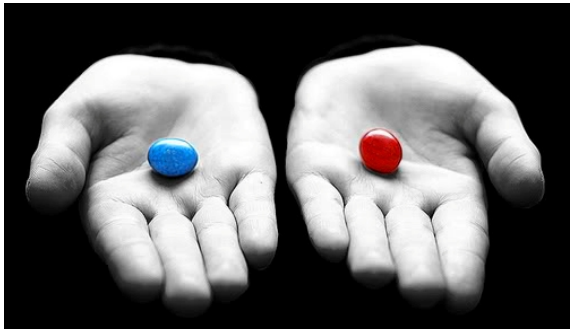
Why to Use an Android Emulator

- Develop Android apps without using a device
- Test apps across multiple Android versions fast
- Automation
- Perform dynamic analysis of suspicious apps
  - Isolation – controlled environment
  - Fast reversion of the environment to a clean state
  - A lot of such tools online

Can a Program Evade Dynamic Analysis?

- Evasive malware
  - Can understand if it is running on a VM or a physical system
  - Hides its malicious behavior when running on a VM

- Pieces of code used to evade dynamic analysis
  - **Red Pill** – able to detect a VM by detecting the hypervisor
  - **Blue Pill** – can shift an OS from the physical computer to a virtualized state under its control



Evading Dynamic Analysis in Android

Category	Type	Examples
Static	Pre-installed static information	IMEI has a fixed value
Dynamic	Dynamic information does not change	Sensors produce always the same value
Hypervisor	VM instruction emulation	Native code runs differently

Static Heuristics

- Device ID (***IdH***)
  - IMEI, IMSI
- Current build (***buildH***)
  - Fields: `PRODUCT`, `MODEL`, `DEVICE`
- Routing table (***netH***)
  - virtual router
    address space: 10.0.2/24
  - Emulated network
    IP address: 10.0.2.15

Static Heuristics

- Device ID (***IdH***)
  - IMEI, IMSI
- Current build (***buildH***)
  - Fields: `PRODUCT`, `MODEL`, `DEVICE`
- Routing table (***netH***)
  - virtual router address space: 10.0.2/24
  - Emulated network IP address: 10.0.2.15

Static Heuristics

- Device ID (***IdH***)
  - IMEI, IMSI
- Current build (***buildH***)
  - Fields: `PRODUCT`, `MODEL`, `DEVICE`
- Routing table (***netH***)
  - virtual router address space: 10.0.2/24
  - Emulated network IP address: 10.0.2.15

IMEI	*123456789012347*	*null*

Static Heuristics

- Device ID (***IdH***)
  - IMEI, IMSI
- Current build (***buildH***)
  - Fields: `PRODUCT`, `MODEL`, `DEVICE`
- Routing table (***netH***)
  - virtual router address space: 10.0.2/24
  - Emulated network IP address: 10.0.2.15

IMEI	*123456789012347*	*null*

Static Heuristics

- Device ID (***IdH***)
  - IMEI, IMSI
- Current build (***buildH***)
  - Fields: `PRODUCT`, `MODEL`, `DEVICE`
- Routing table (***netH***)
  - virtual router address space: 10.0.2/24
  - Emulated network IP address: 10.0.2.15

IMEI	*123456789012347*	*null*
DEVICE	*Nexus*	*generic*

Static Heuristics

- Device ID (***IdH***)
  - IMEI, IMSI
- Current build (***buildH***)
  - Fields: `PRODUCT`, `MODEL`, `DEVICE`
- Routing table (***netH***)
  - virtual router address space: 10.0.2/24
  - Emulated network IP address: 10.0.2.15

IMEI	`123456789012347`	`null`
DEVICE	`Nexus`	`generic`

Static Heuristics

- Device ID (***IdH***)
  - IMEI, IMSI
- Current build (***buildH***)
  - Fields: `PRODUCT`, `MODEL`, `DEVICE`
- Routing table (***netH***)
  - virtual router address space: 10.0.2/24
  - Emulated network IP address: 10.0.2.15

IMEI	*12345678901234*7	*null*
DEVICE	*Nexus*	*generic*
`/proc/net/tcp`	*Ordinary network*	*Emulated network*

Dynamic Heuristics (1/2)

GPS

Accelerometer **Gyroscope**

Gravity Sensor **Proximity Sensor**

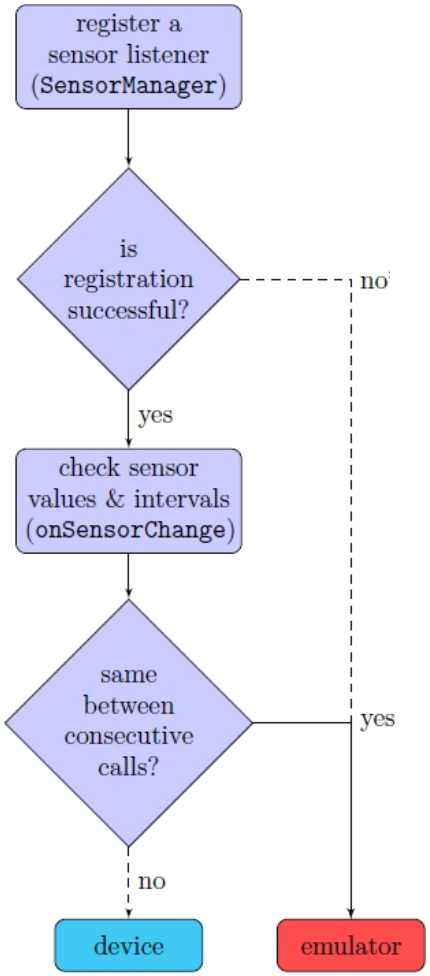
Rotation Vector **Magnetic Field**

Sensors:

- A key difference between mobile & conventional systems
- new opportunities for mobile devices identification
- ***Can emulators realistically simulate device sensors?***
  - Partially: same value, equal time intervals

Dynamic Heuristics (2/2)

- Sensor-based heuristics
- Android Activity that monitors sensors' output values
- We implemented this algorithm for a variety of sensors
  - Accelerometer (**accelH**)
  - magnetic field (**magnFH**)
  - rotation vector (**rotVecH**),
  - proximity (**proximH**)
  - gyroscope (**gyrosH**)



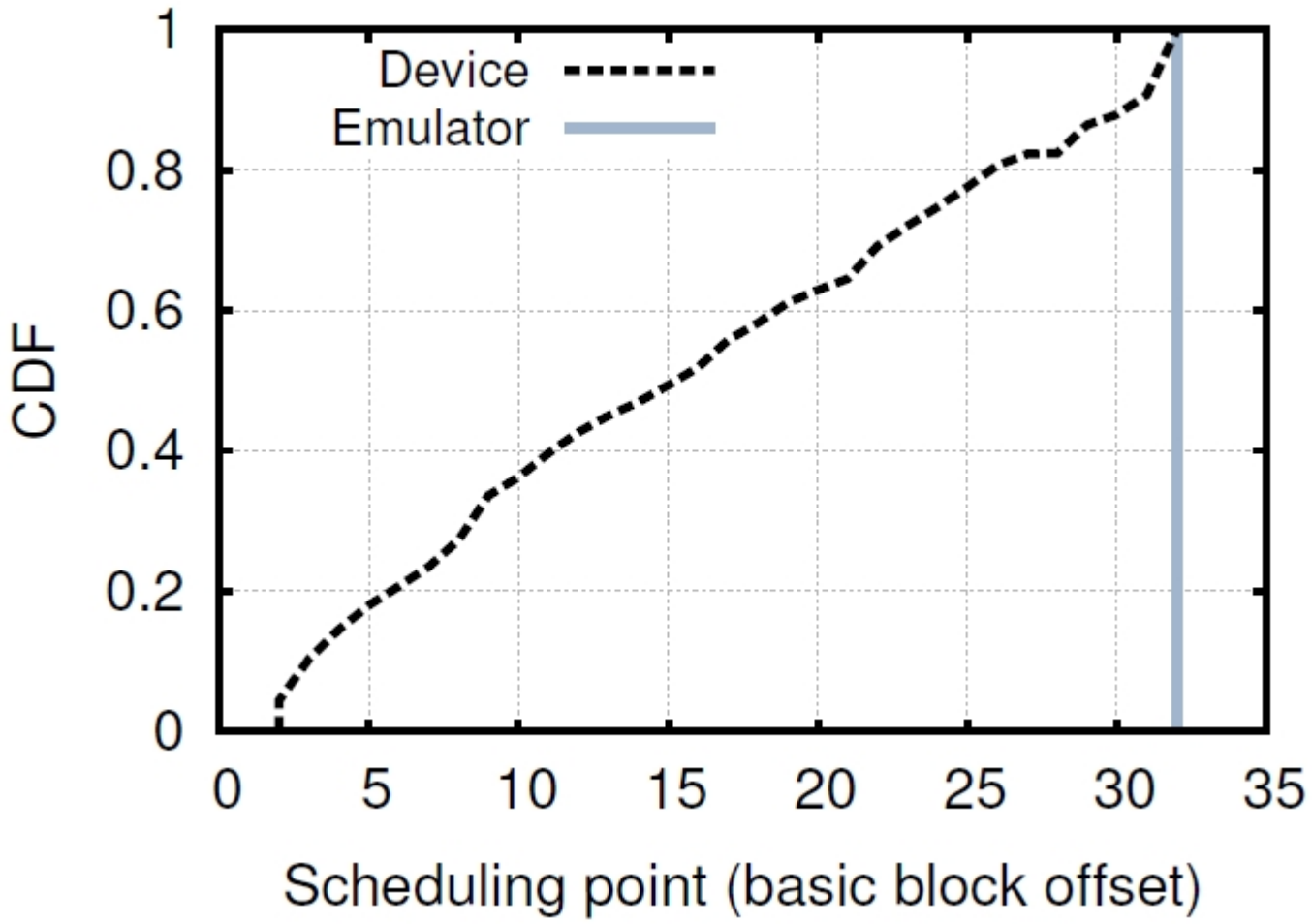
Hypervisor Heuristics

- Try to identify the virtual machine monitor
- Android Emulator is based on **QEMU**
- Heuristics
  - Based on QEMU's incomplete emulation of the actual hardware
  - Identify QEMU scheduling
  - Identify QEMU caching behavior

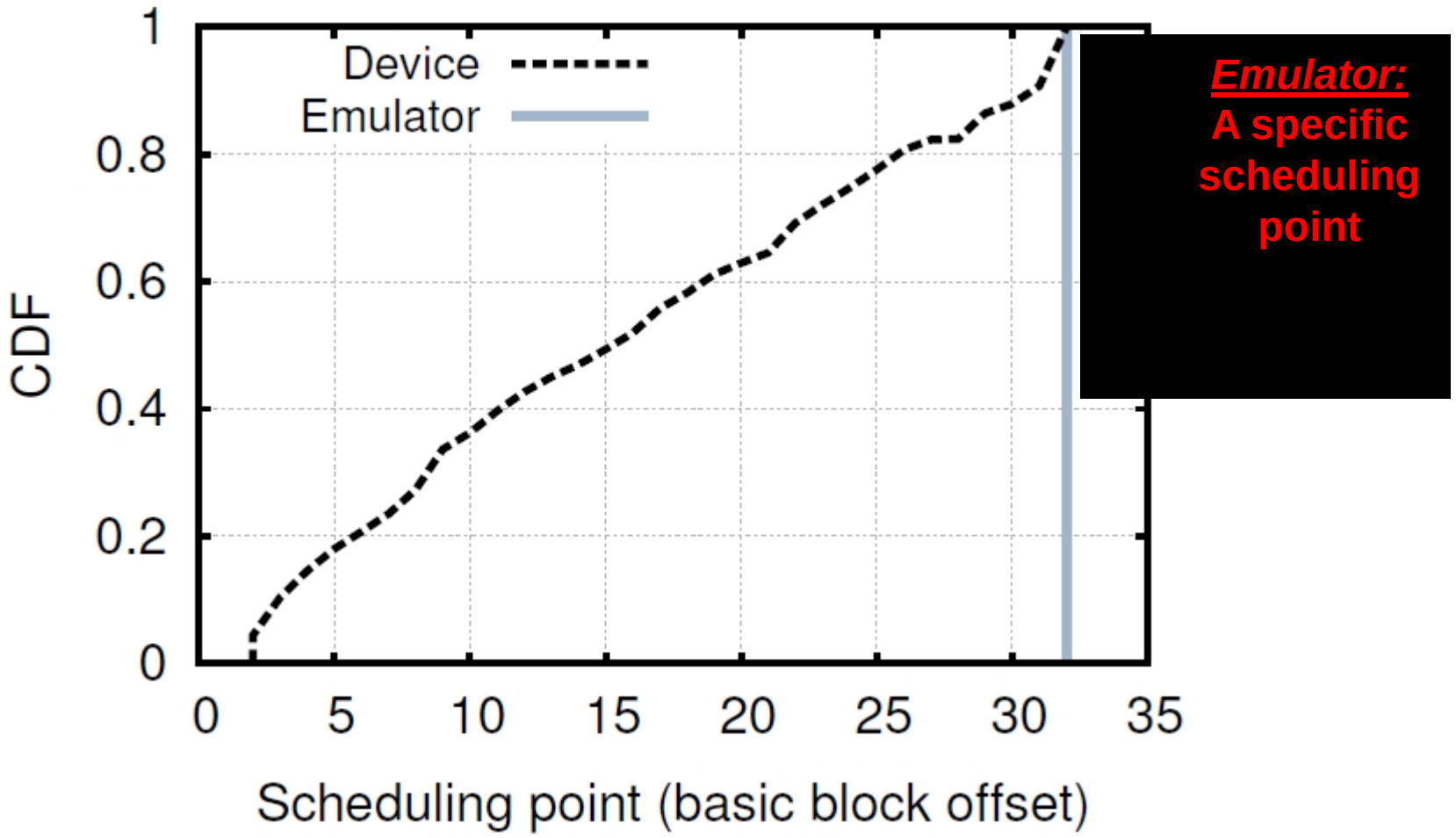
Identify QEMU Scheduling (1/2)

- Virtual PC in QEMU
  - is updated only after the execution of a basic block (**branch**)
  - OS scheduling does not occur during a basic block
- QEMU Binary Translation (BT) Detection
  - Monitor scheduling addresses of a thread
    - Real Device: Various scheduling points
    - Emulator: A unique scheduling point
  - **BTdetectH**

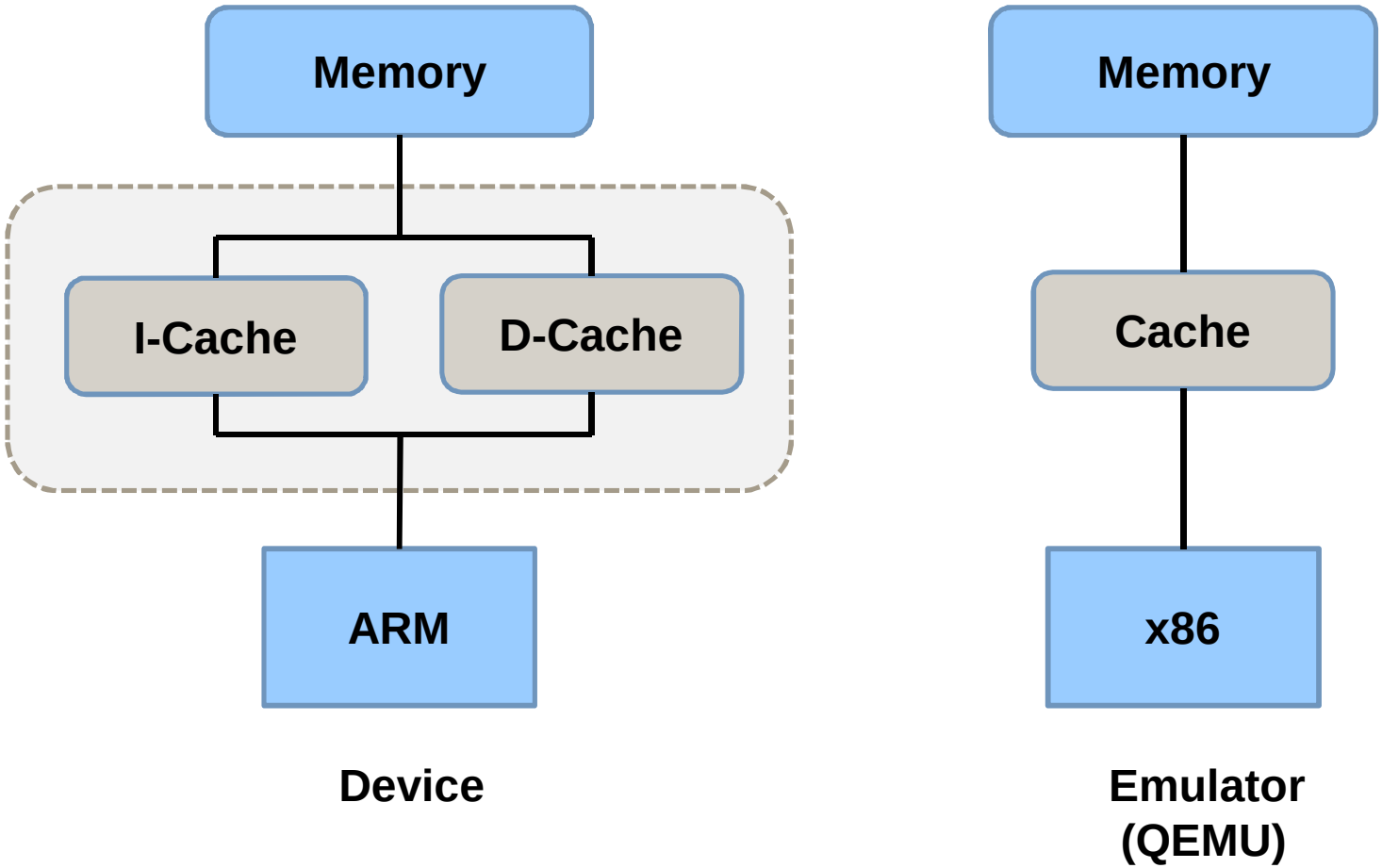
Identify QEMU Scheduling (2/2)



Identify QEMU Scheduling (2/2)



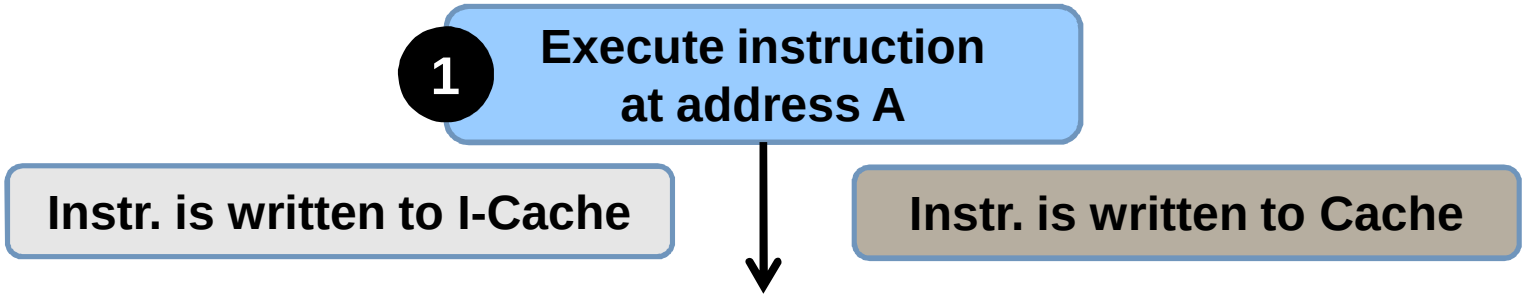
Identify QEMU Caching Behavior (1/2)



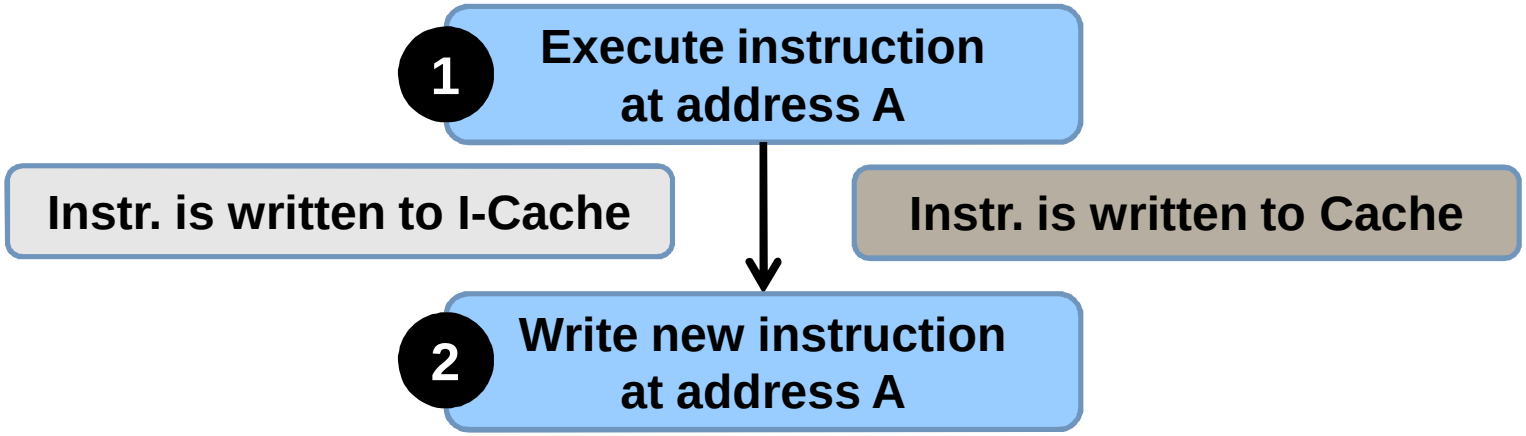
Identify QEMU Caching Behavior (2/2)

1 **Execute instruction at address A**

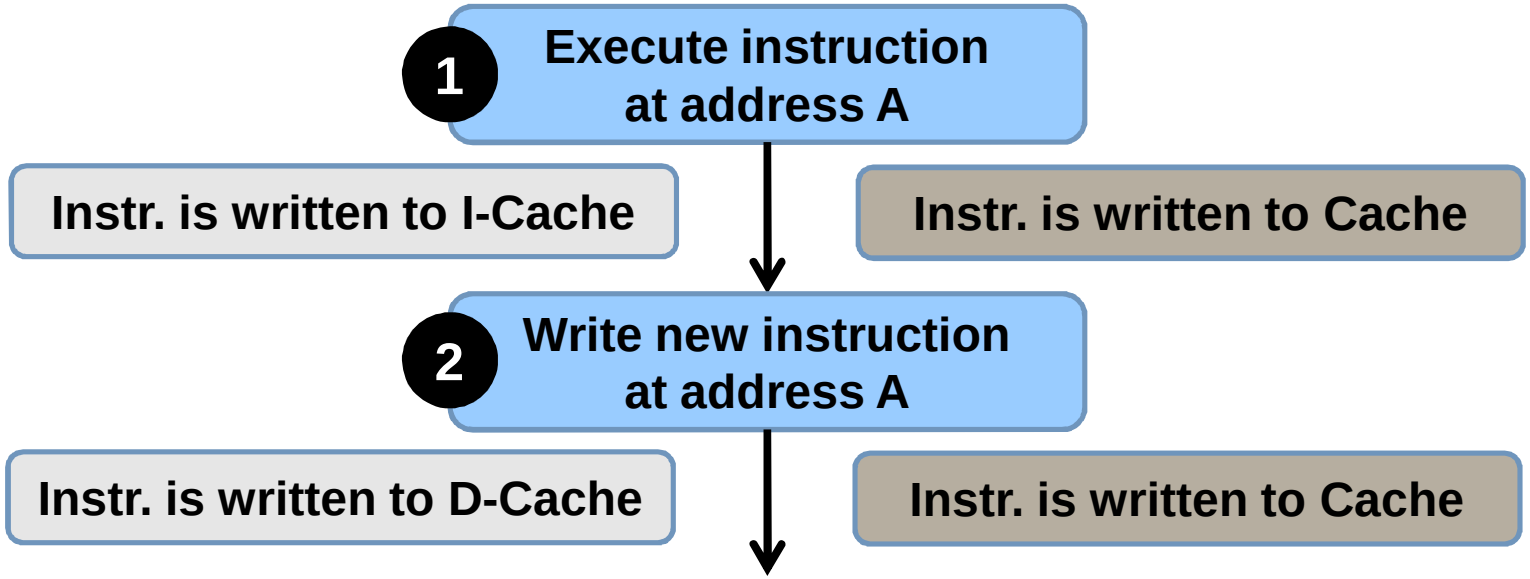
Identify QEMU Caching Behavior (2/2)



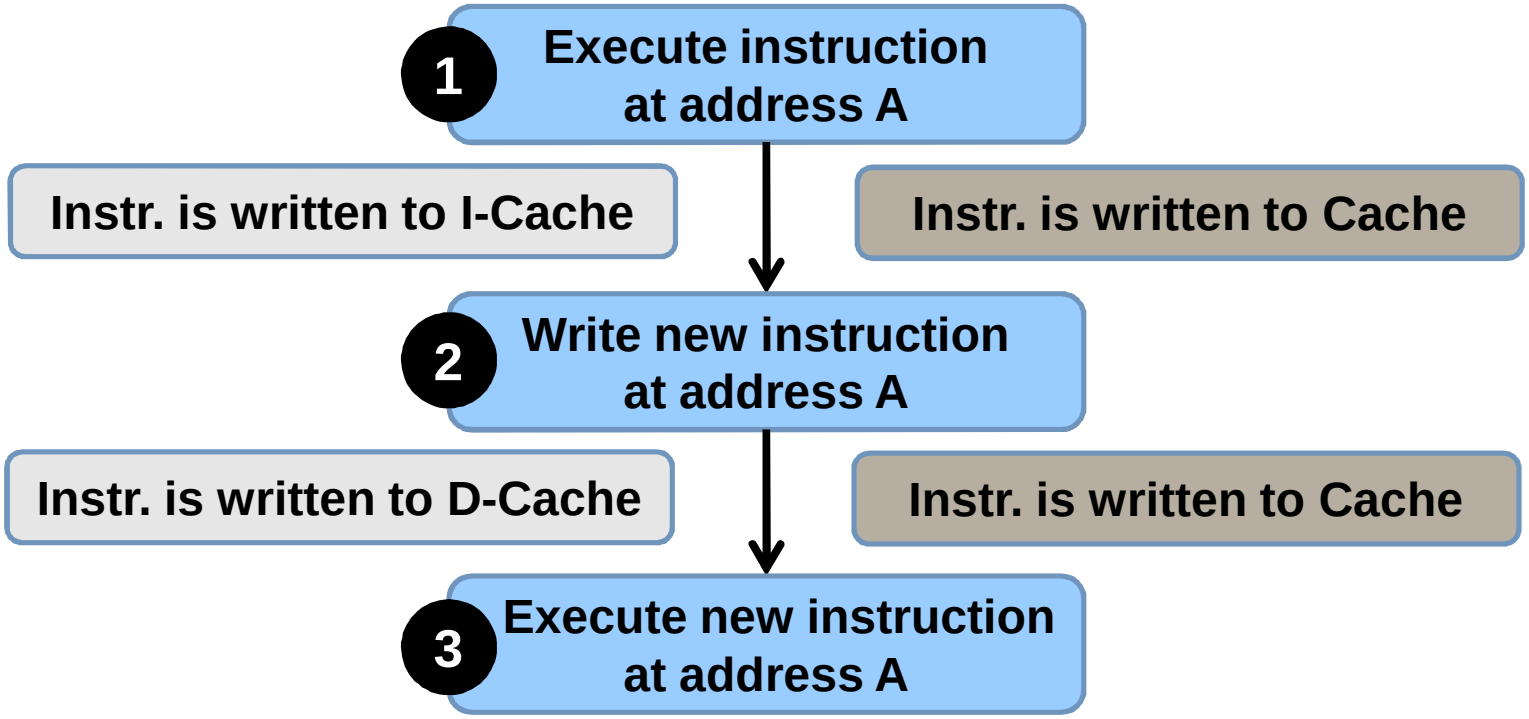
Identify QEMU Caching Behavior (2/2)



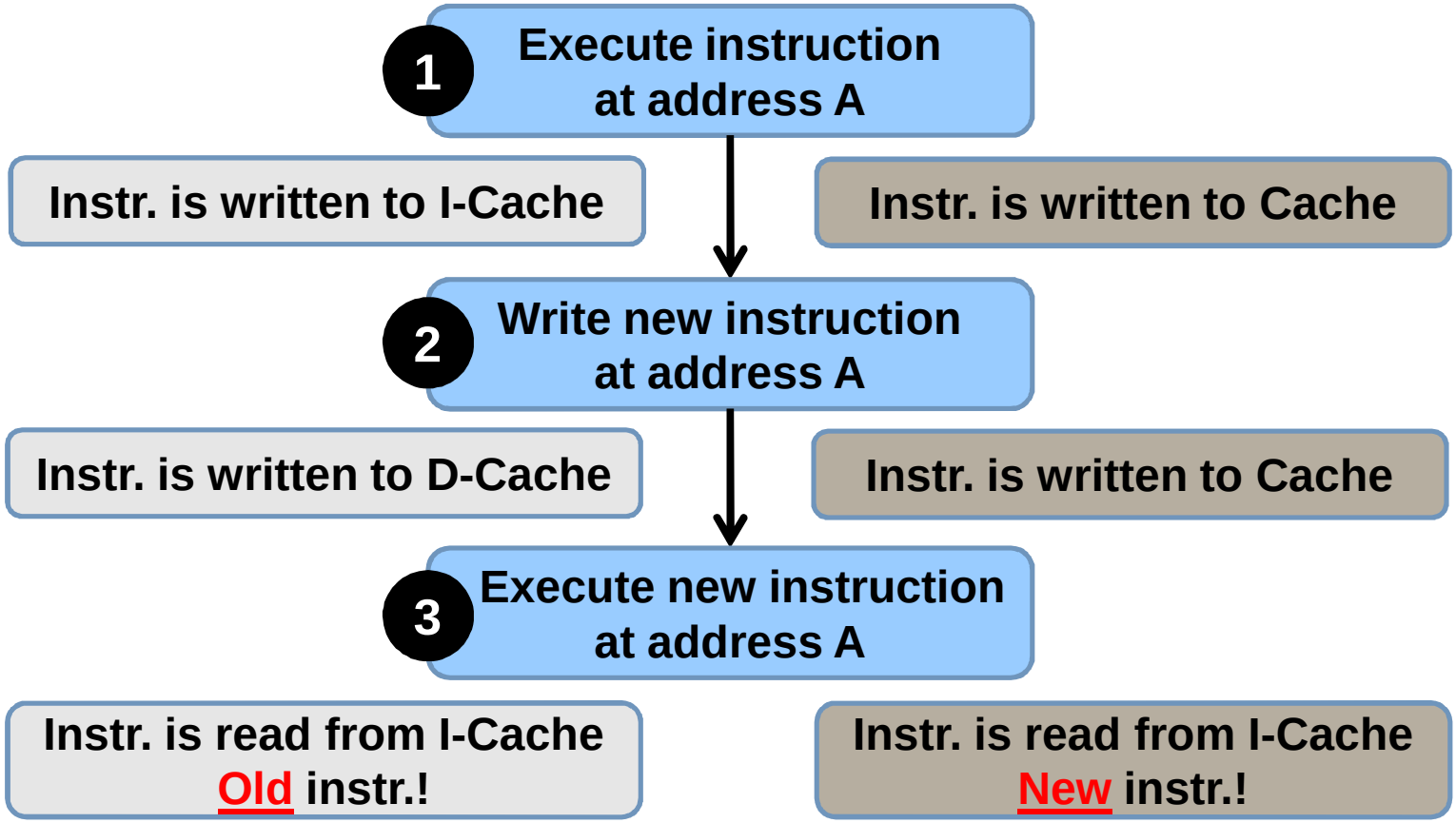
Identify QEMU Caching Behavior (2/2)



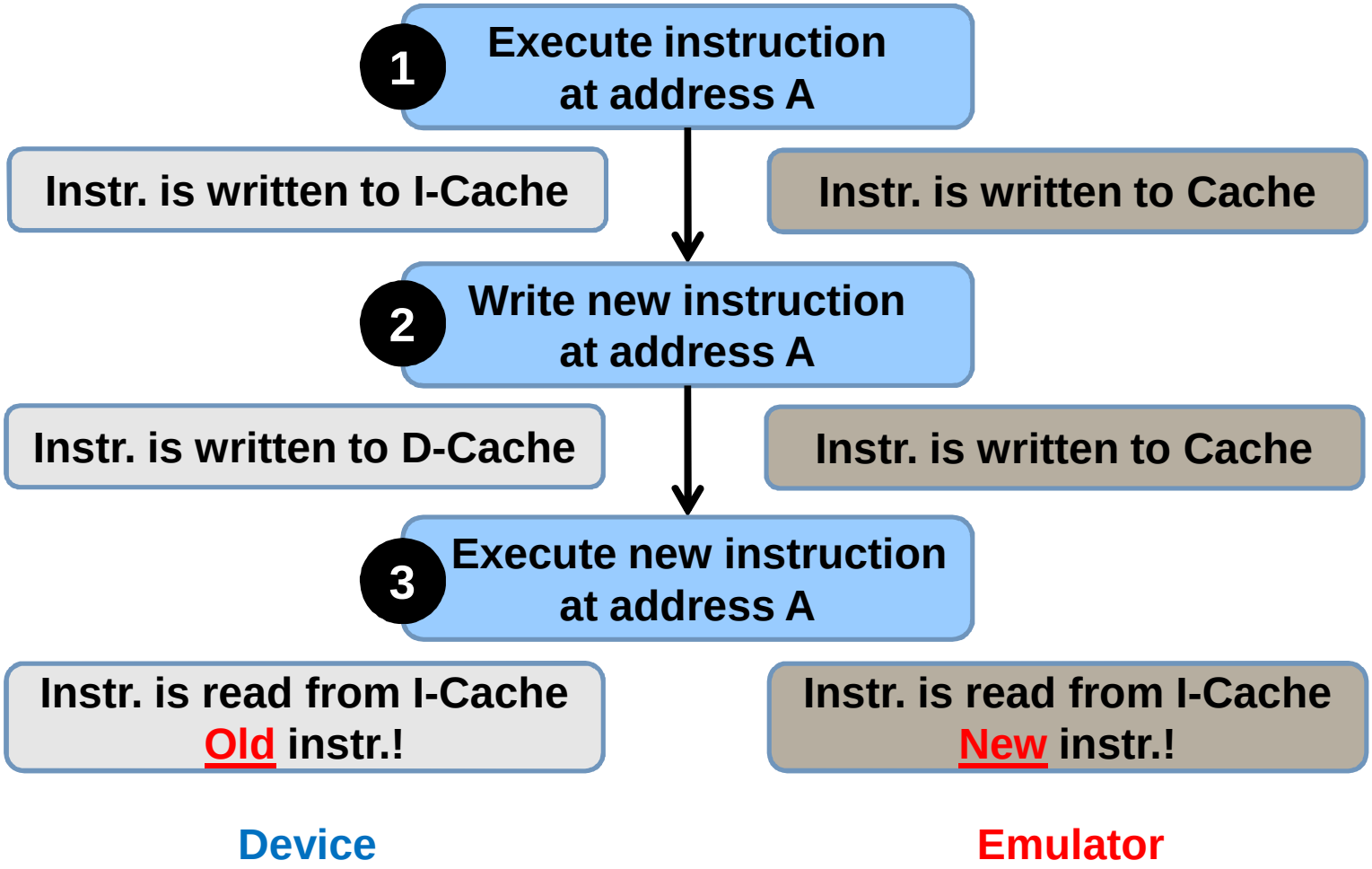
Identify QEMU Caching Behavior (2/2)



Identify QEMU Caching Behavior (2/2)



Identify QEMU Caching Behavior (2/2)



Resilience of dynamic analysis tools

	idH	buildH	netH	accelH	magnFH	rotVecH	proximH	gyrosH	BTdetectH	xFlowH
DroidBox	✓	✗	✗	✗	✗	✗	✗	✗	*JNI NS*	*JNI NS*
DroidScope	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
TaintDroid	✗	✗	✗	✗	✗	✗	✗	✗	*JNI NS*	*JNI NS*
Andrubis	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗
SandDroid	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗
ApkScan	✓	✗	✗	✗	✗	✗	✗	✗	*JNI NS*	*JNI NS*
VisualThreat	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
Tracedroid	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
CopperDroid	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
Apk Analyzer	✓	✓	✓	✗	✗	✗	✗	✗	*JNI NS*	*JNI NS*
ForeSafe	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
Mobile Sandbox	✓	✗	✗	✗	✗	✗	✗	✗	*JNI NS*	*JNI NS*

Countermeasures

- Static heuristics
  - Emulator modifications
- Dynamic heuristics
  - Realistic sensor event simulation
- Hypervisor heuristics
  - Accurate binary translation
  - Hardware-assisted virtualization
  - Hybrid application execution

More about Emulation Evasion

- Check the papers

Rage Against the Virtual Machine: Hindering Dynamic Analysis of Android Malware (Eurosec 2014)

Evading Android Runtime Analysis via Sandbox Detection (ASIACCS 2014)

Next

- *How can we enhance an Android malware with Emulation Evasion capabilities without haven't access to the source code?*