TaintART: A Practical Multi-level Information-Flow Tracking System for Android RunTime

Mingshen Sun, Tao Wei, John C.S. Lui

Sudeep Nanjappa Jayakumar

Agenda

- Android Basics
- Introduction
- Contributions
- SDK Downloads – Google
- Background
- Environments
- Comparison – Android Dalvik & ART Environment
- System Design - TaintART
- Taint tag Storage
- Taint Propagation Logic
- Implementation
- Case Study
- Macrobenchmarks and Microbenchmarks
- Comparison of instruction numbers for different types
- Limitations & related work

Android Basics

What is Android?

- Free, open source mobile platform
  - Source code at http://source.android.com
- Any handset manufacturer or hobbyist can customize
- Any developer can use
  - SDK at http://developer.android.com

Background

Android Overview:

- Android OS is based on the Linux Kernel.
- Android has middleware called application framework which is based on database and App runtime libraries.
- The application framework provides various APIs for apps developers - activity management, content management, and view system.
- Android apps are mainly written in java, but to enhance the performance, developers can embed C/C++ and use Java Native Interface (JNI) to interact with apps and framework APIs.
- Each app runs in an isolated environment. Apps can also communicate with other apps and services through a specific inter-process communication mechanism called the binder.

Introduction

- TaintDroid were designed for the legacy Dalvik environment used for Dynamic taint analysis for Android apps.
- It customizes Android runtime (Dalvik Virtual Machine) to achieve taint storage and taint propagation.
- Latest Android version no longer support TaintDroid because of the compatibility and performance issues.
- TaintART – Dynamic multi level information flow tracking system.
- Supports the latest Android runtime environments.
- TaintART utilizes processor registers for taint storage. Compared to TaintDroid which needs at least two memory accesses
- Multi-level taint analysis technique to minimize the taint tag storage.
- Multi level privacy enforcement is done to protect sensitive data from leakage.

Contributions

- **Methodology:**

Efficiently track dynamic information flows on the Android mobile operating system with ahead-of-time compilation strategy. Here the multi level analysis is done on the optimized code than doing on the original bytecode of the application.

- **Implementation:**

TaintART is implemented on Android Marshmallow. TaintART can track multilevel information flows within the method, across the method and also data transmitted between the different apps.

Contributions Contd…

- **Performance:**

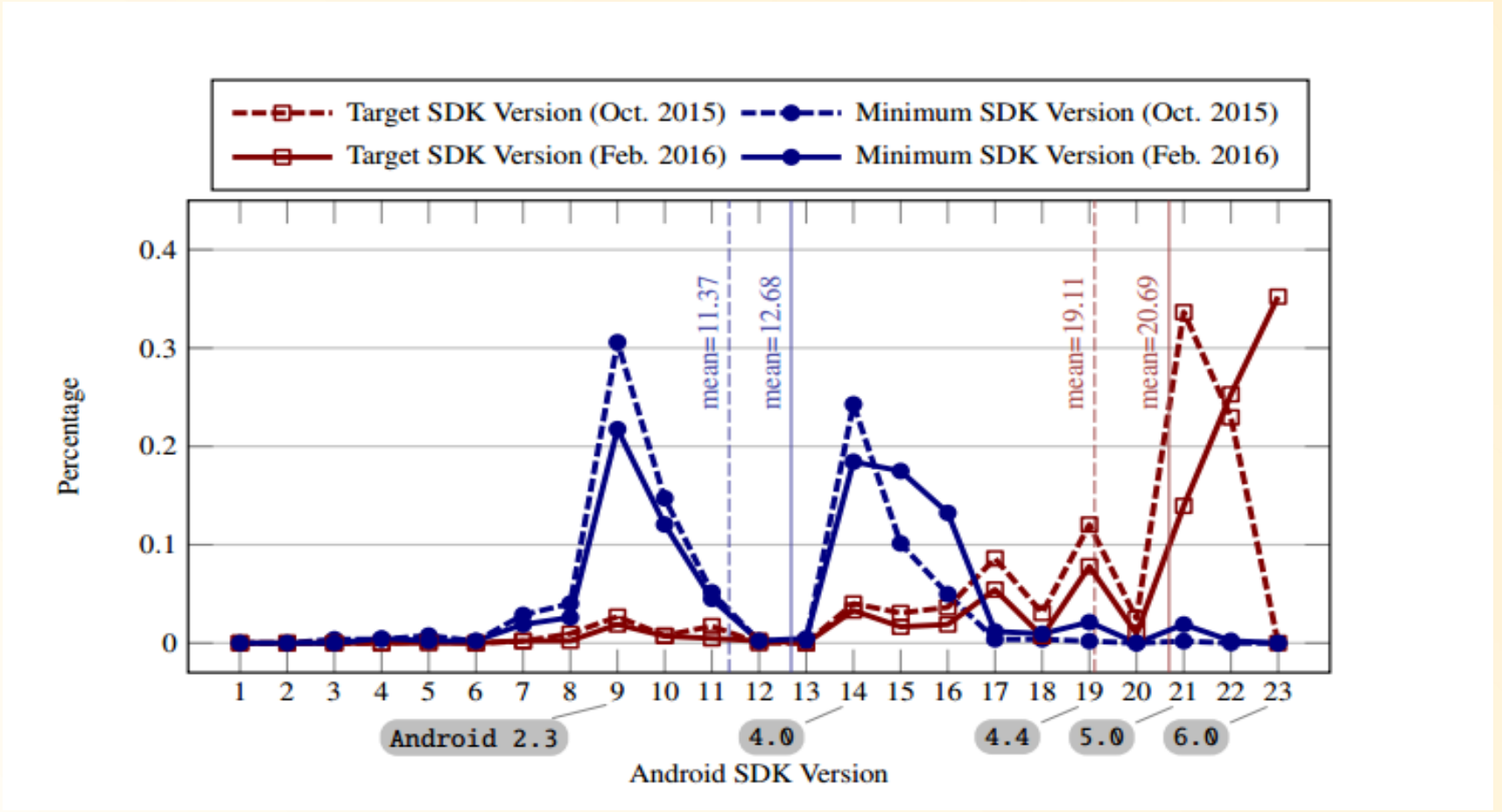
Macrobenchmarks, microbenchmarks and compatibility test are performed on the TaintART. It also achieves 2.5 % and 99.7 % faster for overall performance compared to quick compiler backend ART runtime and Dalvik VM in Android 4.4.

TaintART can analyze apps without compatibility issues.

- **Application to privacy leakage analysis:**

Privacy leakage issues have been addressed on the popular apps in Android 6.0.

SDK Downloads - Google



Environments

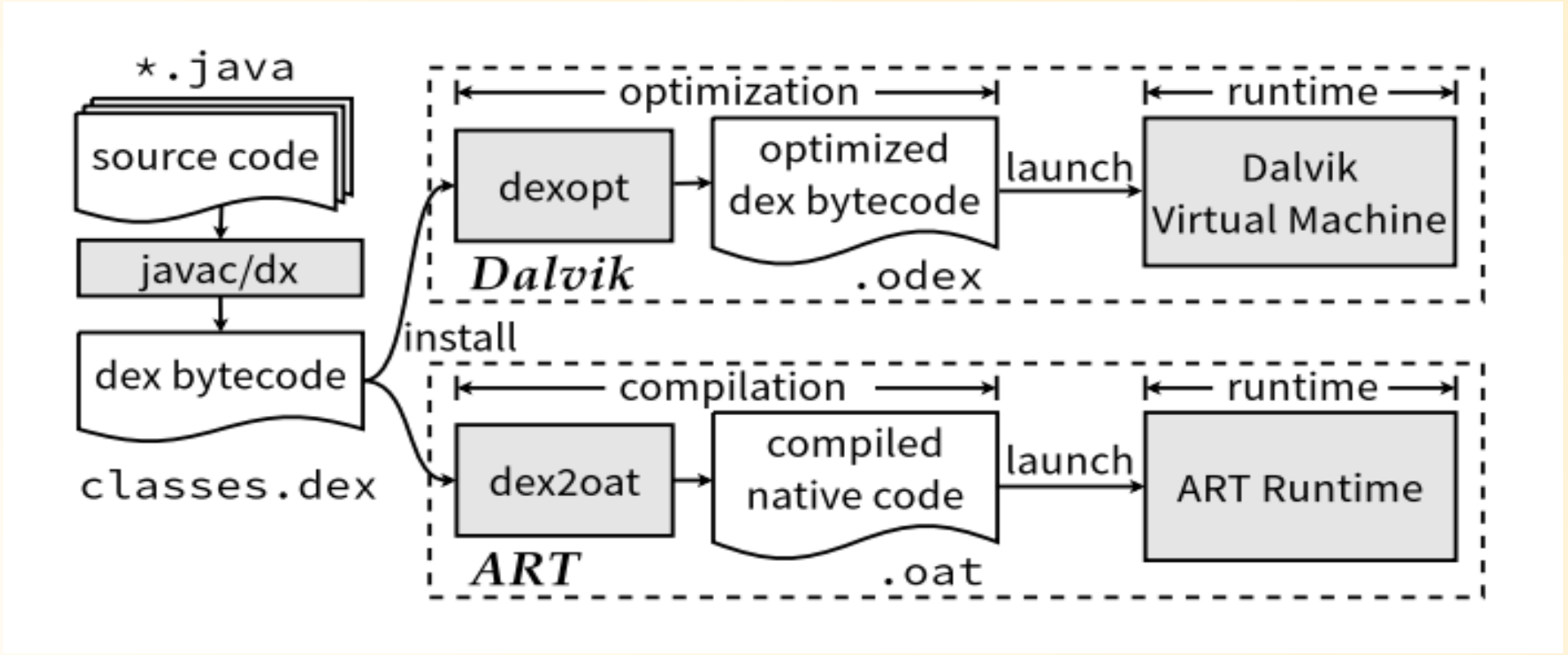
1. Dalvik Environment:

- Dalvik adopts virtual machine interpretation strategy at runtime.
- Dexopt tool will optimize original dex bytecode and at runtime, Dalvik virtual machine will interpret bytecode and execute architecture specific native code.
- Dalvik VM maintains an internal stack for local variables and arguments.

2. ART Enviroment:

- First introduced as experimental environment with Android 4.4
- Replaced Dalvik and was made as default environment
- ART adopts ahead-of-time (AOT) compilation strategy instead of virtual machine interpretation.
- dex2oat tool will directly compile dex bytecode into native code during app's installation and then store as an oat file.
- Dex2oat compiler performs multiple times to achieve better performance.

Comparison – Android Dalvik & ART Environment

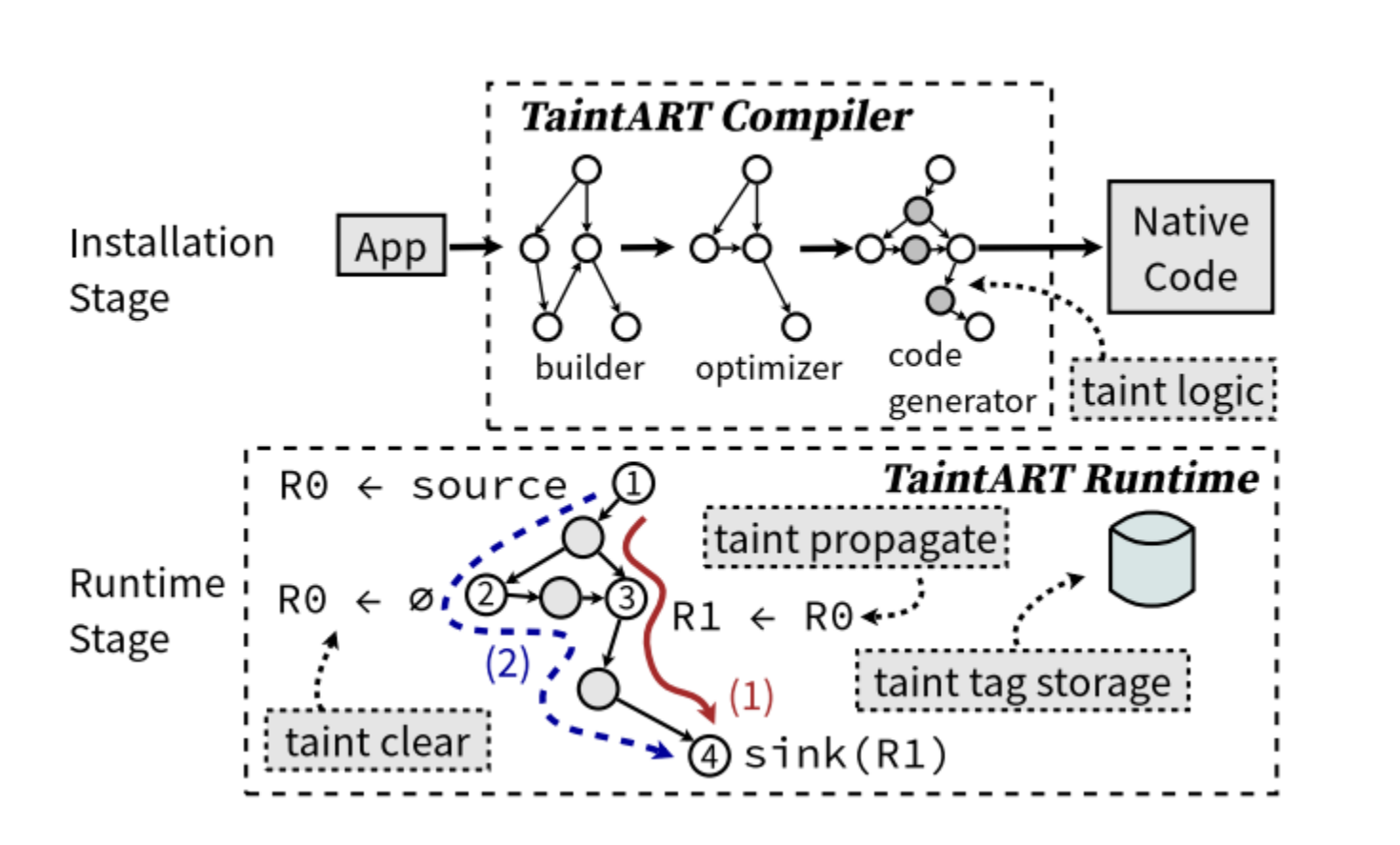


System Design - TaintART

- TaintART utilizes dynamic taint analysis technique and can track data by inserting tracking logic.
- TaintART employs a multi-level taint tag methodology to minimize taint storage so that tags can be stored in processor registers for fast access.
- ART compiler is customized to retain the original ahead of time organizations.
- TaintART's multilevel data tracking strategy is used for policy enforcement on data leakage.
- In dynamic taint analysis, sensitive data is targeted at any sensitive function called **taint source** and **taint tag** will be labeled on the sensitive data for tracking.
- When the data is copied or transformed to another place, its taint tag will **propagate** to the new place.

System Design - TaintART

- The taint tag status for tracking data will be stored in **taint tag storage**.
- If any tainted data leaves the system at some specified functions called **taint sinks.**



Taint tag Storage

- Built on Google Nexus 5 – 32 bit ARM platform.
- 16 CPU registers, each with 32 bits.
- Register R5 is reserved for taint storage .
- Register allocator of TaintART will ensure R5 is not assigned for other purposes such as variable storage.
- First sixteen bits (from bit 0 to bit 15) will be used for storing taint tags of sixteen registers (from R0 to R15).
- The remaining sixteen bits are used for storing taint tag of floating point registers (from S0 to S15).

Taint Propagation Logic

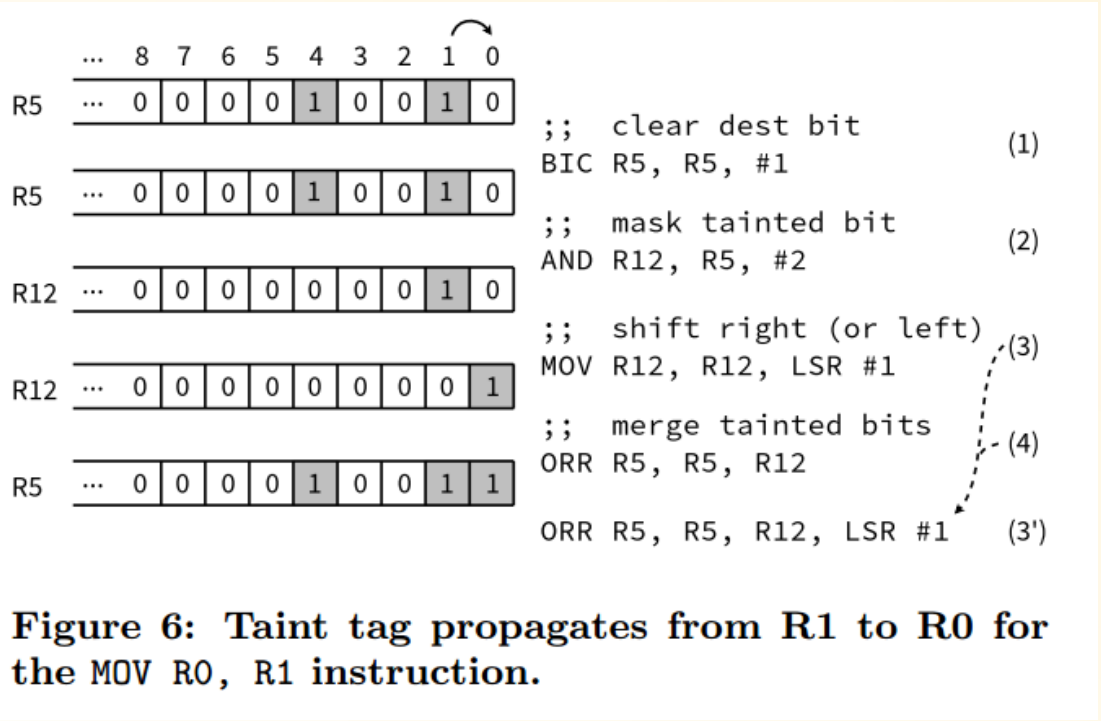


Figure 6: Taint tag propagates from R1 to R0 for the `MOV R0, R1` instruction.

- TaintART introduces much less instructions on handling the taint status changes.
- There are two registers involved R5 as the taint storage register & R12 register for the temporary usage.
- Involves 4 steps: clear destination bit, masking tainted bit, shifting bits, and merging tainted bits.
- TaintART needs only three data processing instructions without memory access to efficiently propagate a taint label.
- This will be good to track the runtime and the performance impacts.

WAYNE STATE

Implementation

Taint sources and sinks:

- TaintART can also be used to enforce policy on sensitive data leakage.
- Four types of data from fifteen sources are tracked and it is categorized in to device identity, sensor data, sensitive content and location data.
- Taint source logic is placed in corresponding classes to track these data.
- When it comes to device identity apps can acquire telephony data by sending the request to telephony manager and in return the taint source logic will attach a tag in the binder parcel.
- location data and sensitive content such as messages, contact lists and call logs are categorized in the third level. These data are considered as level three data and as most sensitive data.

Taint sources and privacy leakage levels

Table 2: Taint Sources and Privacy Leakage Levels

Level	Leaked Data	Source	Class/Service
0 (00)	No Leakage	N/A	N/A
1 (01)	Device Identity	IMSI	`TelephonyManager`
		IMEI	`TelephonyManager`
		ICCID	`TelephonyManager`
		SN	`TelephonyManager`
2 (10)	Sensor Data	Accelerometer	`SensorManager`
		Rotation	`SensorManager`
	Location Data	GPS Location	`LocationManager`
		Last Seen Location	`LocationManager`
		Network Location	`LocationManager`
3 (11)	Sensitive Content	SMS	`ContentResolver`
		MMS	`ContentResolver`
		Contacts	`ContentResolver`
		Call log	`ContentResolver`
		File content	`File`
		Camera	`Camera`
		Microphone	`MediaRecorder`

Implementation

Taint Analysis Interface:

- Two basic interfaces can be developed for taint analysis.
- addTaint() & getTaint() – These can be used to update taint tag of a specific local variables or objects and inspect taint tag later.
- These two inter
- faces are implemented in order to achieve better performance.

Implementation & Deployment

- The prototype of TaintART is implemented on Android 6.0.1 Marshmallow for Nexus 5.
- ART compiler and ART runtime sources are customized to implement taint tag propagation.
- Binder related sources are also customized in Android framework.
- They provide customized binary and libraries such as dex2oat, libart.so and libart-compiler.so
- Since the code base of ART environment is stable after Android 5.0, the implementation is generic for Android 5.0 and 6.0 versions.
- Analysts can overwrite our customized binary and libraries to a target device with root privilege. There is no need of reinstalling the customized systems from scratch.

Case Study

Experimental Setup – TaintDroid is downloaded and compiled which is based on Android 4.3.

- TaintART is run on Android 6.0.1 & apps used in the case study were downloaded from the Google play in May 2016.

Privacy Tracking – Popular apps were tested and potential privacy leakage was checked.

- They manually interacted with each app in TaintDroid and TaintART and recorded the reports of privacy leakage.

Privacy Leakage Analysis

Table 3: Privacy Leakage Analysis on Popular Apps.

App Name	Version	Min/Target SDK	TaintDroid Result (Error Message)	TaintART Result
Taobao	5.7.2	14/23	Some functions are broken: "cannot find method" in config error	2: device identity, sensor data, location data
Alipay	9.6.6.051201	15/23	Cannot login: "It is crowed" error	2: device identity, sensor data, location data
JD.COM	5.1.0	14/14	Device identity and accelerometer leakage	2: device identity, sensor data, location data
Facebook	77.0.0.20.66	21/23	Cannot install: the minimum SDK is Android 5.0	1: device identity
Skype	6.34.0.715	15/23	Device identity leakage	1: device identity
Instagram	8.1.0	16/23	Device identity leakage	1: device identity
Spotify	5.3.0.995	15/23	No leakage	0: no leakage
Amazon Shopping	6.6.0.100	11/23	No leakage	0: no leakage

Case Study

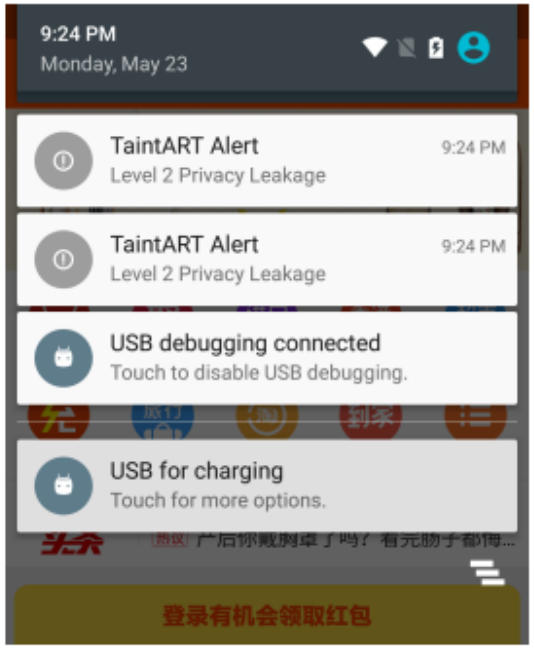
Policy Enforcement – Since TaintARt supports latest Android runtime it is easy to deploy the policy enforcement.

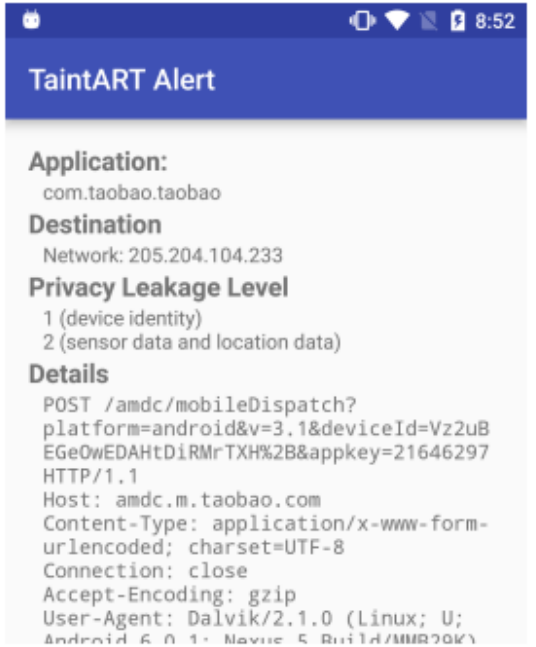
- Here users can pre-define multi-level policy rules.

- For each level users can define different policies.

Table 4: Privacy Enforcement Policy.

Level	Description of Enforcement Policy
0	N/A
1	record events
2	record events, alert users and rewrite sensitive information
3	record events, alert users and prevent accesses





Macrobencmarks

- TaintART is a general framework that can be used by end-users to protect their privacy.
- Several macrobenchmarks were performed to measure the overhead for normal usage of the applications.

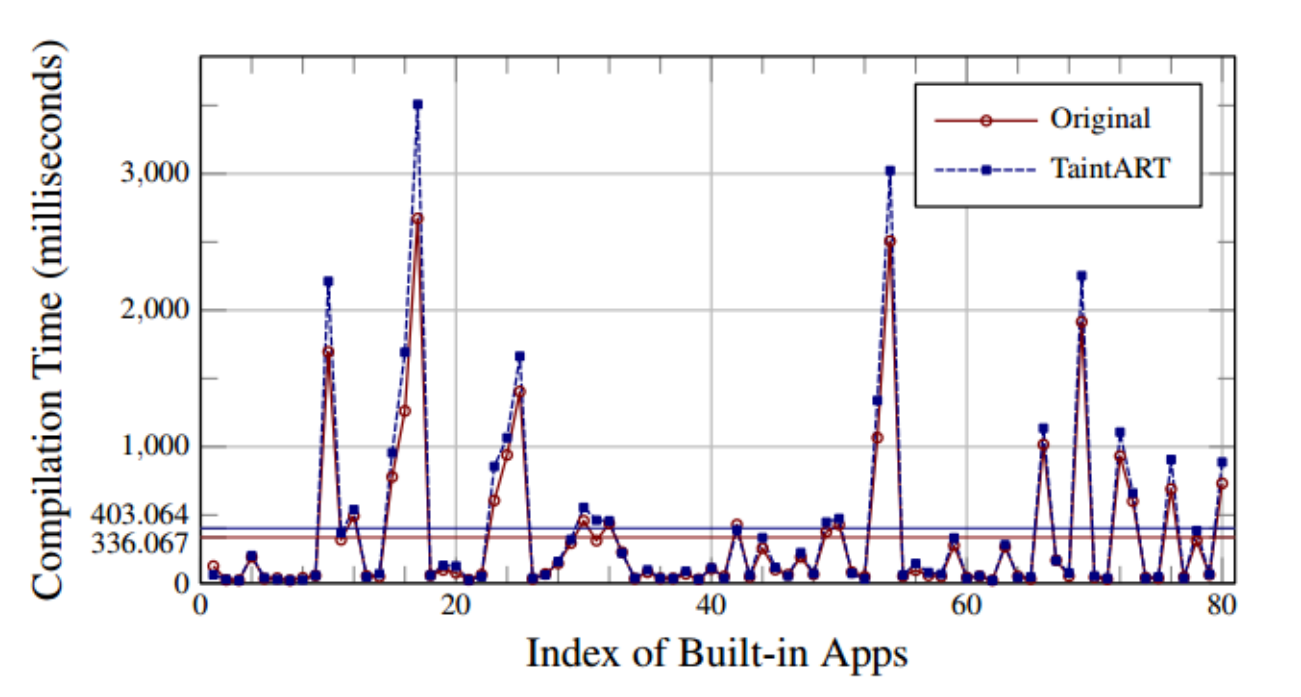
Table 5: Macrobenchmark Results.

Macrobenchmark Name (ms)	Original (with Optimizing Backend)	TaintART
App Launch Time	348.2	370.3
App Installation Time	1680.5	1886.3
Contacts Read/Write	7.0/9538.5	8.4/9655.2

Microbenchmarks

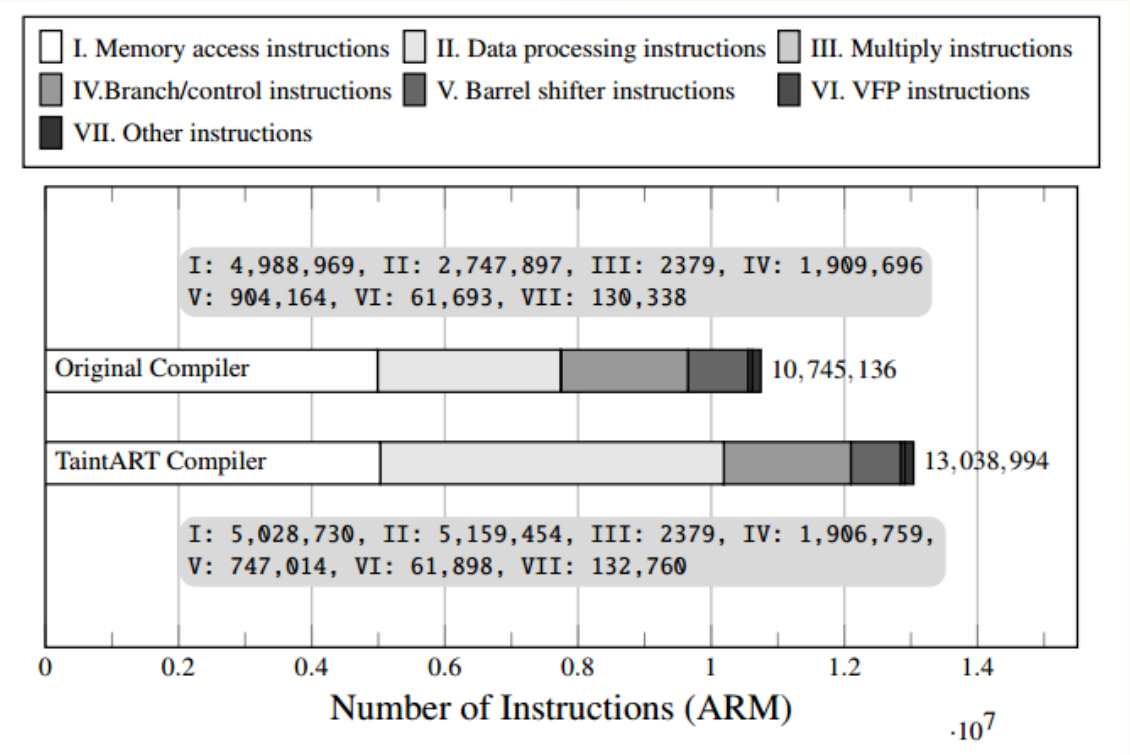
Compiler Benchmarks – By adopting the TaintART the compilation time is increased by 336.076 milliseconds to 403.064 milliseconds and introduces about 19.9 % overhead.

- The below figure illustrates the compilation time for 80 built-in apps.



WAYNE STATE

Comparison of instruction numbers for different types



- The total number of instructions increases about 21 %.
- The increases are mainly in data processing instructions (Type II) including arithmetic instructions (ADD, SUB), logical instructions (ORR, AND), movement instructions (MOV, MVN).
- TaintART compiler only introduces about 0.8 % more instructions.
- This means that TaintART can achieve better runtime performance than the VM-based TaintDroid with the gains of AOT compilation strategy in the new ART environment.

WAYNE STATE

Limitations

- TaintART cannot track specific data flows.
- All implicit leakage cannot be tracked.
- Complex malwares can detect the presence of TaintART and can hide their activities with few some anti analysis techniques to detect host devices.
- Malware analysis, analysts need to manually trigger the behaviors

Related Work

- There are many systems which dynamically monitor the runtime information in different layers of the system and few of them are DroidScope, BareCloud and CopperDroid introspect Dalvik VM to capture dynamic information for reconstructing malware behaviors.
- There are many systems which still use the static analysis system for disassembled code and try to precisely model runtime behavior and use program analysis technique to resolve information flows and few of them are Android Leaks and Flowdroid.
- Also there are many systems to detect suspicious behaviors and prevent potential privacy leakage and few of them are Aurasium and RetroSkeleton which can add enforcement policies and fine-grained mandatory access control on sensitive API invocations by rewriting and repackaging apps.

Thank you