Systems and Internet Infrastructure Security

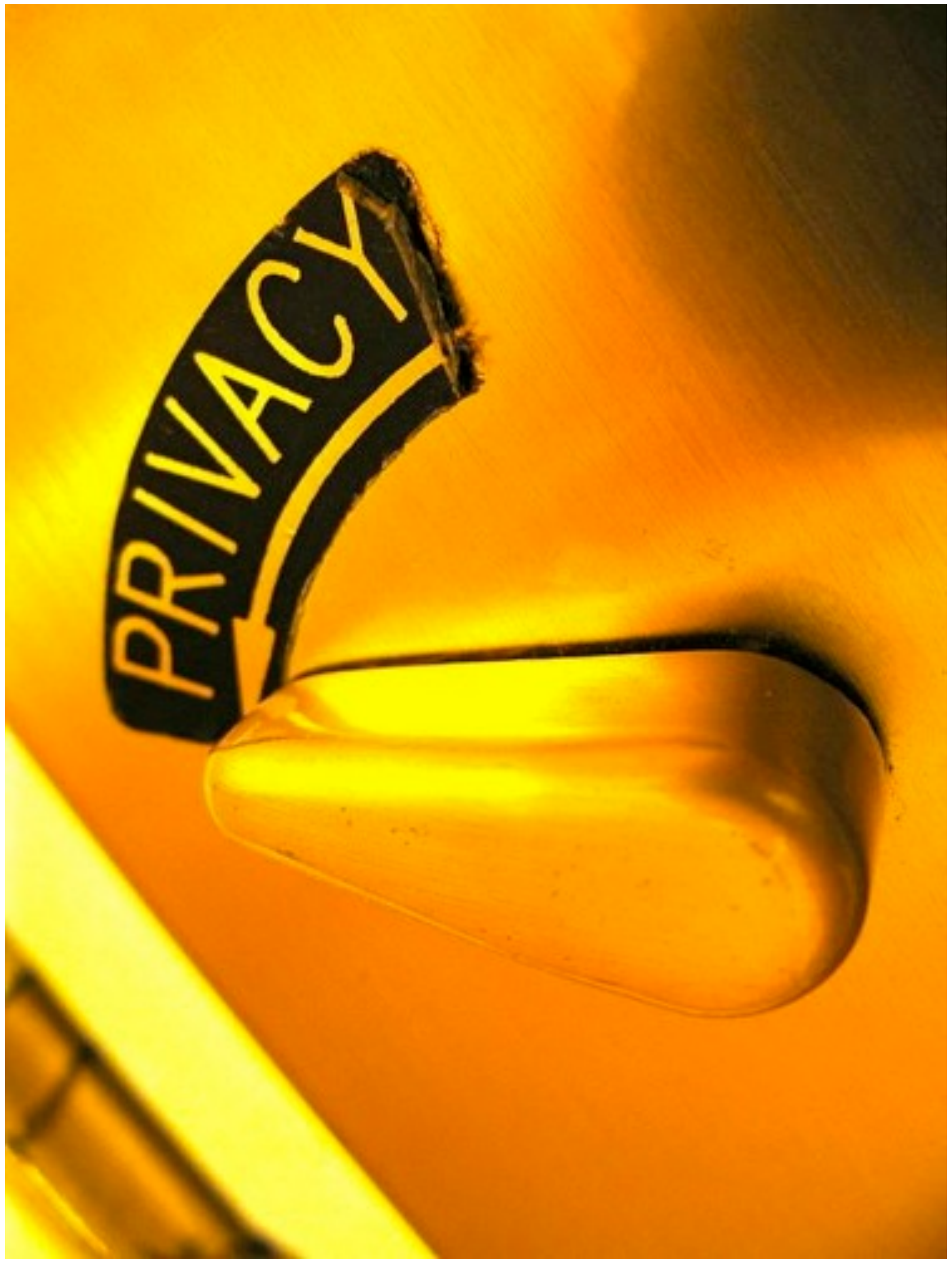
Network and Security Research Center
Department of Computer Science and Engineering
Pennsylvania State University, University Park PA

TaintDroid: An Information-Flow Tracking System for Realtime Privacy Monitoring on Smartphones

OSDI'10

William Enck, Peter Gilbert, Byung-Gon Chun, Landon P. Cox, Jaeyeon Jung, Patrick McDaniel, and Anmol N. Sheth

Smartphone Privacy?



(http://www.flickr.com/photos/pong/2404940312/)

Monitoring Smartphone Behavior

- There are tens of thousands of smartphone apps that provide both fun and valuable utility.
- *General challenge*: balance fun and utility with privacy
- Step 1: "look inside" of applications to watch how they use privacy sensitive data
  - location
  - phone identifiers
  - microphone
  - camera
  - address book

Challenges

- *Goal*: Monitor app behavior to determine when privacy sensitive information leaves the phone

- *Challenges …*
  - *Smartphones are resource constrained*
  - *Third-party applications are entrusted with several types of privacy sensitive information*
  - *Context-based privacy information is dynamic and can be difficult to identify even when sent in the clear*
  - *Applications can share information*

Dynamic Taint Analysis

PENNSTATE 1855

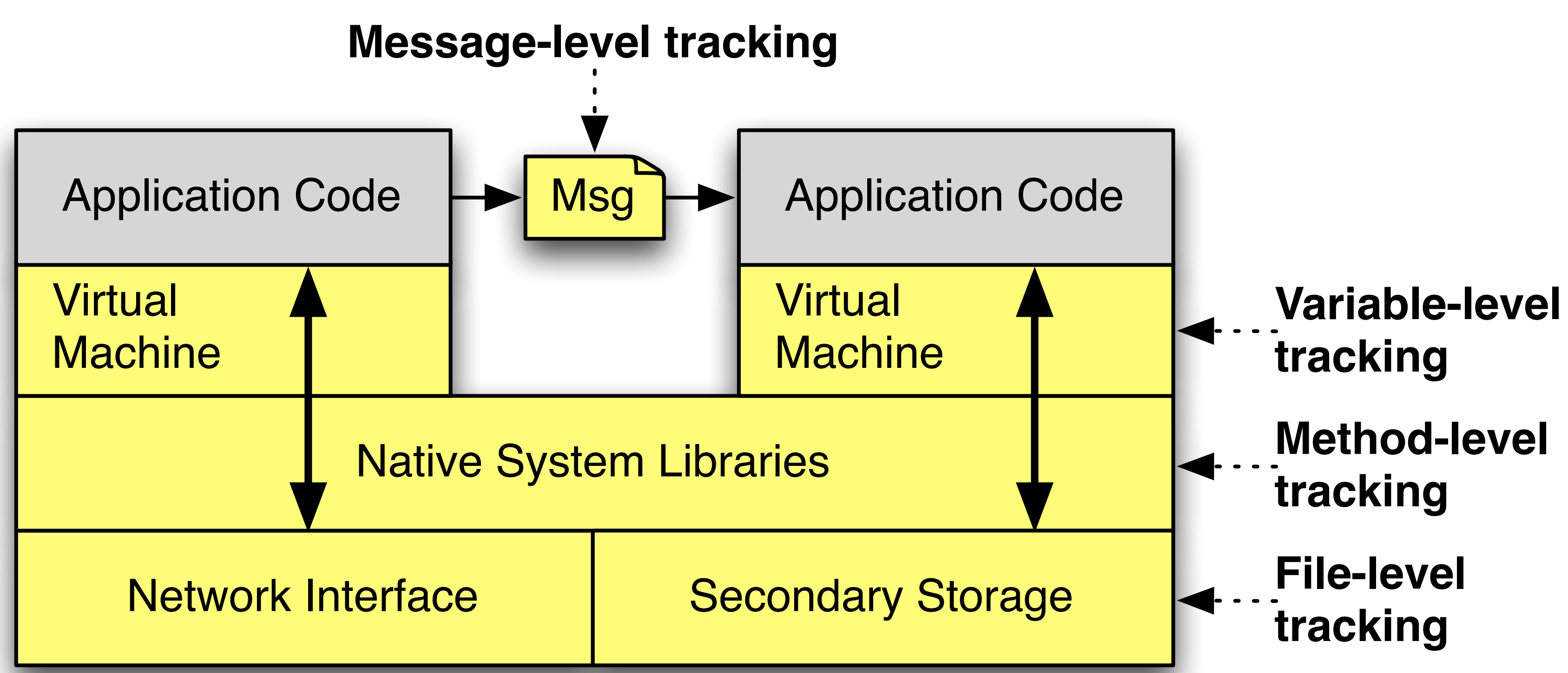
- Dynamic taint analysis is a technique that tracks information dependencies from an origin
- Conceptual idea:
  - Taint source
  - Taint propagation
  - Taint sink

```
c = taint_source()
...
a = b + c
...
network_send(a)
```

- *Limitations*: performance and granularity is a trade-off

TaintDroid

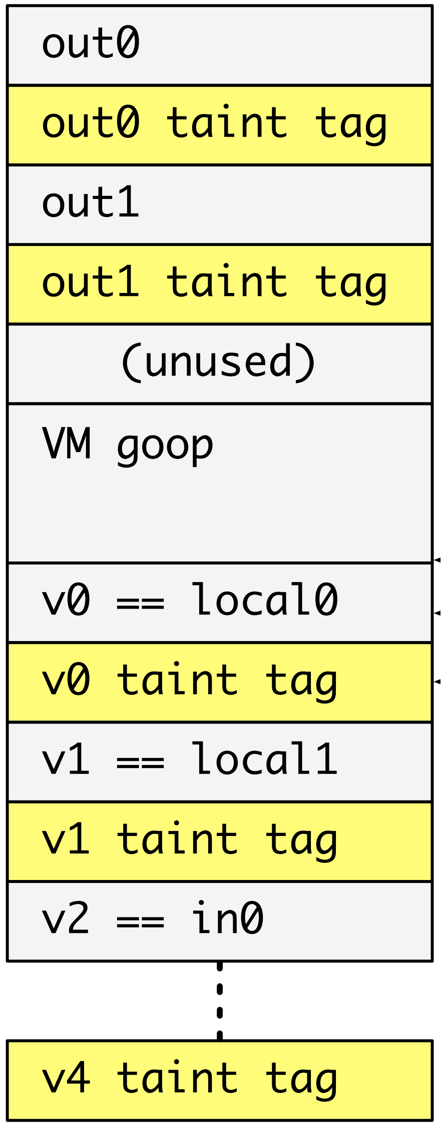
- TaintDroid is a system-wide integration of taint tracking into the Android platform
  - Variable tracking throughout Dalvik VM environment
  - Patches state after native method invocation
  - Extends tracking between applications and to storage



- *TaintDroid is a firmware modification, not an app*

VM Variable-level Tracking

- We modified the Dalvik VM interpreter to *store* and *propagate* taint tags (a taint bit-vector) on variables.
- *Local variables and args*: taint tags stored adjacent to variables on the internal execution stack.
    - 64-bit variables span 32-bit storage
- *Class fields*: similar to locals, but inside static and instance field heap objects
- *Arrays*: one taint tag per array to minimize overhead



DEX Propagation Logic

- *Data flow*: propagate source regs to destination reg

Op Format	Op Semantics	Taint Propagation	Description
const-op v_A C	$v_A \leftarrow C$	$\tau(v_A) \leftarrow \emptyset$	Clear v_A taint
move-op v_A v_B	$v_A \leftarrow v_B$	$\tau(v_A) \leftarrow \tau(v_B)$	Set v_A taint to v_B taint
move-op-R v_A	$v_A \leftarrow R$	$\tau(v_A) \leftarrow \tau(R)$	Set v_A taint to return taint
return-op v_A	$R \leftarrow v_A$	$\tau(R) \leftarrow \tau(v_A)$	Set return taint ($\emptyset$ if void)
move-op-E v_A	$v_A \leftarrow E$	$\tau(v_A) \leftarrow \tau(E)$	Set v_A taint to exception taint
throw-op v_A	$E \leftarrow v_A$	$\tau(E) \leftarrow \tau(v_A)$	Set exception taint
unary-op v_A v_B	$v_A \leftarrow \otimes v_B$	$\tau(v_A) \leftarrow \tau(v_B)$	Set v_A taint to v_B taint
binary-op v_A v_B v_C	$v_A \leftarrow v_B \otimes v_C$	$\tau(v_A) \leftarrow \tau(v_B) \cup \tau(v_C)$	Set v_A taint to v_B taint $\cup$ v_C taint
binary-op v_A v_B	$v_A \leftarrow v_A \otimes v_B$	$\tau(v_A) \leftarrow \tau(v_A) \cup \tau(v_B)$	Update v_A taint with v_B taint
binary-op v_A v_B C	$v_A \leftarrow v_B \otimes C$	$\tau(v_A) \leftarrow \tau(v_B)$	Set v_A taint to v_B taint
aput-op v_A v_B v_C	$v_B[v_C] \leftarrow v_A$	$\tau(v_B[\cdot]) \leftarrow \tau(v_B[\cdot]) \cup \tau(v_A)$	Update array v_B taint with v_A taint
aget-op v_A v_B v_C	$v_A \leftarrow v_B[v_C]$	$\tau(v_A) \leftarrow \tau(v_B[\cdot]) \cup \tau(v_C)$	Set v_A taint to array and index taint
sput-op v_A f_B	$f_B \leftarrow v_A$	$\tau(f_B) \leftarrow \tau(v_A)$	Set field f_B taint to v_A taint
sget-op v_A f_B	$v_A \leftarrow f_B$	$\tau(v_A) \leftarrow \tau(f_B)$	Set v_A taint to field f_B taint
iput-op v_A v_B f_C	$v_B(f_C) \leftarrow v_A$	$\tau(v_B(f_C)) \leftarrow \tau(v_A)$	Set field f_C taint to v_A taint
iget-op v_A v_B f_C	$v_A \leftarrow v_B(f_C)$	$\tau(v_A) \leftarrow \tau(v_B(f_C)) \cup \tau(v_B)$	Set v_A taint to field f_C and object reference taint

DEX Propagation Logic

PENNSTATE

1855

- *Data flow*: propagate source regs to destination reg

Op Format	Op Semantics	Taint Propagation	Description
const-op v_A C	$v_A \leftarrow C$	$\tau(v_A) \leftarrow \emptyset$	Clear v_A taint
move-op v_A v_B	$v_A \leftarrow v_B$	$\tau(v_A) \leftarrow \tau(v_B)$	Set v_A taint to v_B taint
move-op-R v_A	$v_A \leftarrow R$	$\tau(v_A) \leftarrow \tau(R)$	Set v_A taint to return taint
return-op v_A	$R \leftarrow v_A$	$\tau(R) \leftarrow \tau(v_A)$	Set return taint ($\emptyset$ if void)
move-op-E v_A	$v_A \leftarrow E$	$\tau(v_A) \leftarrow \tau(E)$	Set v_A taint to exception taint
throw-op v_A	$E \leftarrow v_A$	$\tau(E) \leftarrow \tau(v_A)$	Set exception taint
unary-op v_A v_B	$v_A \leftarrow \otimes v_B$	$\tau(v_A) \leftarrow \tau(v_B)$	Set v_A taint to v_B taint
binary			
binary			
binary			
aput-op v_A v_B v_C	$v_B[v_C] \leftarrow v_A$	$\tau(v_B[\cdot]) \leftarrow \tau(v_B[\cdot]) \cup \tau(v_A)$	Update array v_B taint with v_A taint
aget-op v_A v_B v_C	$v_A \leftarrow v_B[v_C]$	$\tau(v_A) \leftarrow \tau(v_B[\cdot]) \cup \tau(v_C)$	Set v_A taint to array and index taint
sput-op v_A f_B	$f_B \leftarrow v_A$	$\tau(f_B) \leftarrow \tau(v_A)$	Set field f_B taint to v_A taint
sget-op v_A f_B	$v_A \leftarrow f_B$	$\tau(v_A) \leftarrow \tau(f_B)$	Set v_A taint to field f_B taint
iput-op v_A v_B f_C	$v_B(f_C) \leftarrow v_A$	$\tau(v_B(f_C)) \leftarrow \tau(v_A)$	Set field f_C taint to v_A taint
iget-op v_A v_B f_C	$v_A \leftarrow v_B(f_C)$	$\tau(v_A) \leftarrow \tau(v_B(f_C)) \cup \tau(v_B)$	Set v_A taint to field f_C and object reference taint

aget-op v_A v_B v_C $\qquad v_A \leftarrow v_B[v_C] \qquad \tau(v_A) \leftarrow \tau(v_B[\cdot]) \cup \tau(v_C)$

DEX Propagation Logic

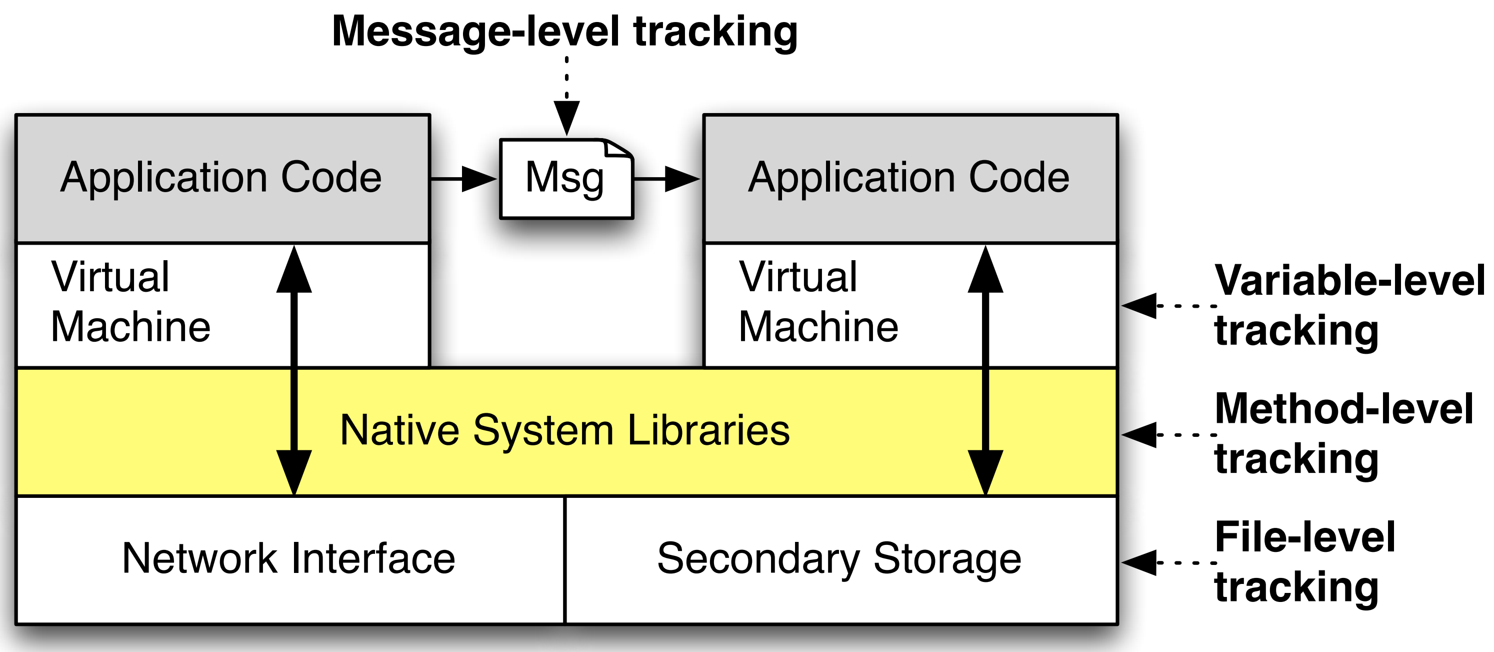
- *Data flow*: propagate source regs to destination reg

Op Format	Op Semantics	Taint Propagation	Description
const-op v_A C	$v_A \leftarrow C$	$\tau(v_A) \leftarrow \emptyset$	Clear v_A taint
move-op v_A v_B	$v_A \leftarrow v_B$	$\tau(v_A) \leftarrow \tau(v_B)$	Set v_A taint to v_B taint
move-op-R v_A	$v_A \leftarrow R$	$\tau(v_A) \leftarrow \tau(R)$	Set v_A taint to return taint
return-op v_A	$R \leftarrow v_A$	$\tau(R) \leftarrow \tau(v_A)$	Set return taint ($\emptyset$ if void)
move-op-E v_A	$v_A \leftarrow E$	$\tau(v_A) \leftarrow \tau(E)$	Set v_A taint to exception taint
throw-op v_A	$E \leftarrow v_A$	$\tau(E) \leftarrow \tau(v_A)$	Set exception taint
unary-op v_A v_B	$v_A \leftarrow \otimes v_B$	$\tau(v_A) \leftarrow \tau(v_B)$	Set v_A taint to v_B taint
binary-op v_A v_B v_C	$v_A \leftarrow v_B \otimes v_C$	$\tau(v_A) \leftarrow \tau(v_B) \cup \tau(v_C)$	Set v_A taint to v_B taint $\cup$ v_C taint
bi[illegible]	[illegible]	[illegible]	[illegible]
bi[illegible]	[illegible]	[illegible]	[illegible]
aput-op v_A v_B v_C	$v_B[v_C] \leftarrow v_A$	$\tau(v_B[\cdot]) \leftarrow \tau(v_B[\cdot]) \cup \tau(v_A)$	Update array v_B taint with v_A taint
aget-op v_A v_B v_C	$v_A \leftarrow v_B[v_C]$	$\tau(v_A) \leftarrow \tau(v_B[\cdot]) \cup \tau(v_C)$	Set v_A taint to array and index taint
sput-op v_A f_B	$f_B \leftarrow v_A$	$\tau(f_B) \leftarrow \tau(v_A)$	Set field f_B taint to v_A taint
sget-op v_A f_B	$v_A \leftarrow f_B$	$\tau(v_A) \leftarrow \tau(f_B)$	Set v_A taint to field f_B taint
iput-op v_A v_B f_C	$v_B(f_C) \leftarrow v_A$	$\tau(v_B(f_C)) \leftarrow \tau(v_A)$	Set field f_C taint to v_A taint
iget-op v_A v_B f_C	$v_A \leftarrow v_B(f_C)$	$\tau(v_A) \leftarrow \tau(v_B(f_C)) \cup \tau(v_B)$	Set v_A taint to field f_C and object reference taint

iget-op v_A v_B f_C $\quad v_A \leftarrow v_B(f_C) \quad \tau(v_A) \leftarrow \tau(v_B(f_C)) \cup \tau(v_B)$

Native Methods

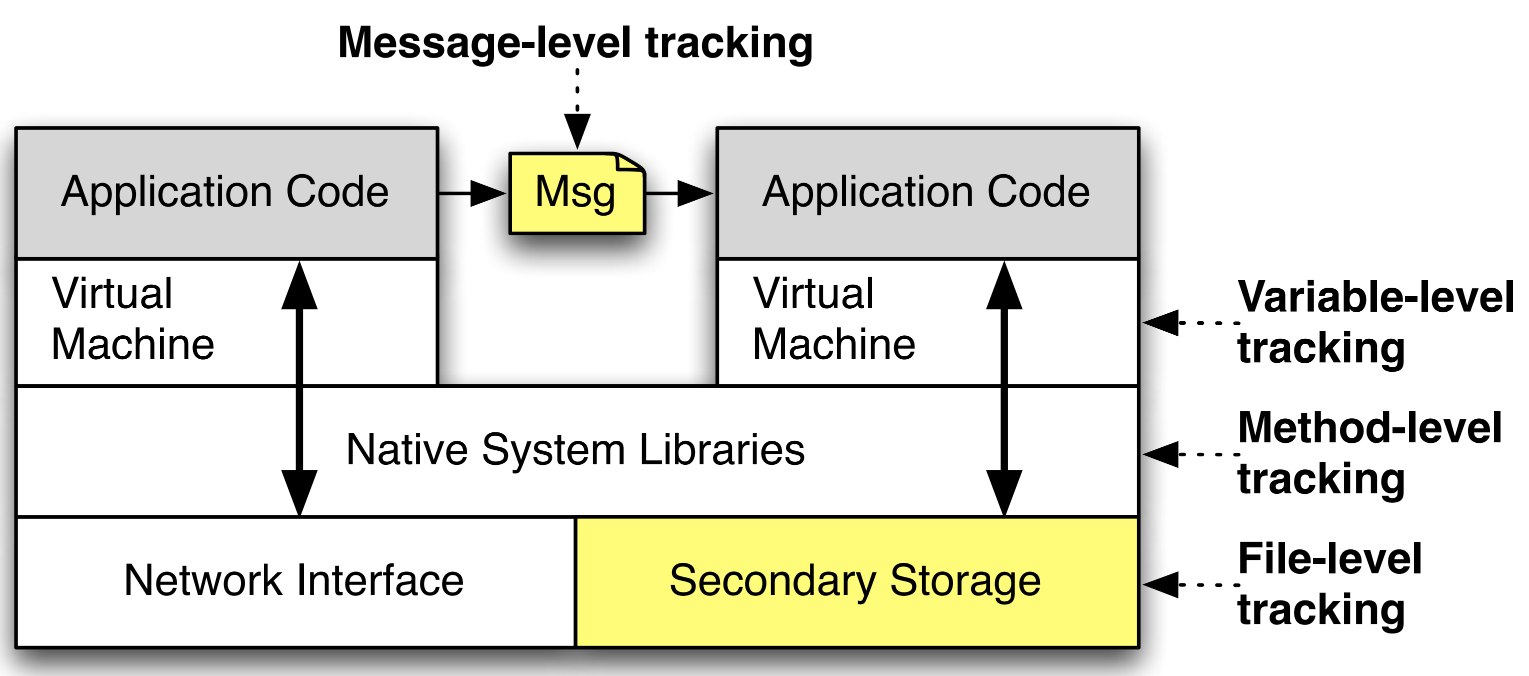
- Applications execute *native methods* through the Java Native Interface (JNI)
- TaintDroid uses a combination of heuristics and *method profiles* to patch VM tracking state
  - Applications are restricted to only invoking native methods in system-provided libraries



IPC and File Propagation

PENNSTATE 1855

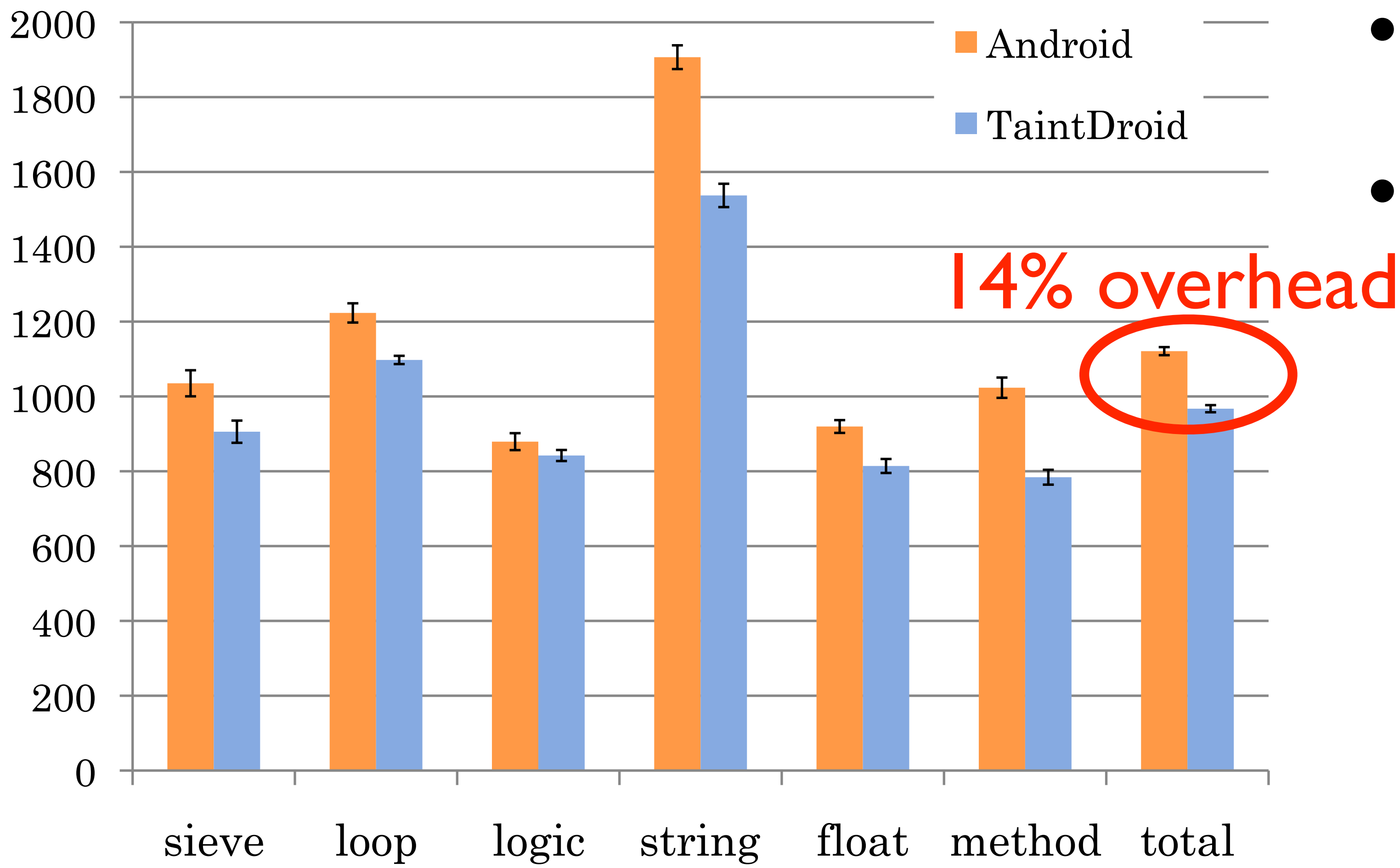
- TaintDroid uses *message level* tracking for IPC
  - Applications marshall and unmarshall individual data items
- Persistent storage tracked at the *file level*
  - Single taint tag stored in the file system XATTR



Performance

CaffeineMark 3.0 benchmark

(higher is better)



CaffeineMark score roughly corresponds to the number of Java instructions per second.

- Memory overhead: 4.4%
- IPC overhead: 27%
- Macro-benchmark:
  - App load: 3% (2ms)
  - Address book: (< 20 ms) 5.5% create, 18% read
  - Phone call: 10% (10ms)
  - Take picture: 29% (0.5s)

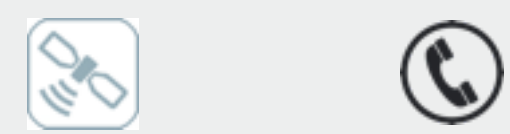
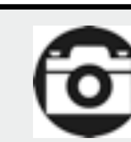
Taint Adaptors

- Taint *sources* and *sinks* must be carefully integrated into the existing architectural framework.

- Depends on information properties
  - *Low-bandwidth sensors*: location, accelerometer
  - *High-bandwidth sensors*: microphone, camera
  - *Information databases*: address book, SMS storage
  - *Device identifiers*: IMEI, IMSI*, ICC-ID, Ph. #
  - *Network taint sink*

Application Study

- Selected 30 applications with bias on popularity and access to *Internet*, *location*, *microphone*, and *camera*

applications	#	permissions
The Weather Channel, Cetos, Solitarie, Movies, Babble, Manga Browser	6	
Bump, Wertago, Antivirus, ABC --- Animals, Traffic Jam, Hearts, Blackjack, Horoscope, 3001 Wisdom Quotes Lite, Yellow Pages, Datelefonbuch, Astrid, BBC News Live Stream, Ringtones	14	
Layer, Knocking, Coupons, Trapster, Spongebot Slide, ProBasketBall	6	
MySpace, Barcode Scanner, ixMAT	3	
Evernote	1	

- *Of 105 flagged connections, only 37 clearly legitimate*

Findings - Location

- 15 of the 30 applications shared physical location with an ad server (*admob.com*, *ad.qwapi.com*, *ads.mobclix.com*, *data.flurry.com*)
- Most traffic was plaintext (e.g., AdMob HTTP GET):

```
...&s=a14a4a93f1e4c68&..&t=062A1CB1D476DE85
B717D9195A6722A9&d%5Bcoord%5D=47.6612278900
00006%2C-122.31589477&...
```

- In no case was sharing obvious to user or in EULA
  - In some cases, periodic and occurred without app use

Findings - Phone Identifiers

PENNSTATE 1855

- 7 applications sent device (*IMEI*) and 2 apps sent phone info (*Ph. #*, *IMSI**, *ICC-ID*) to a remote server without informing the user.
  - One app's EULA indicated the IMEI was sent
  - Another app sent the hash of the IMEI
- Frequency was app-specific, e.g., one app sent phone information every time the phone booted.
- Appeared to be sent to app developers ...

> "There have been cases in the past on other mobile platforms where well-intentioned developers are simply over-zealous in their data gathering, without having malicious intent." -- Lookout

Limitations

- Approach limitations:
  - TaintDroid only tracks data flows (i.e., explicit flows).
- Taint source limitations:
  - IMSI contains country (MCC) and network (MNC) codes
  - File databases must be all one type

Summary

- TaintDroid provides efficient, system-wide, dynamic taint tracking and analysis for Android

- We found 20 of the 30 studied applications to share information in a way that was not expected.

- Source code will be available soon: *appanalysis.org*

- Future investigations:
  - Provide direct feedback to users
  - Potential for realtime enforcement
  - Integration with expert rating systems

Demo

- Demo available at http://appanalysis.org/demo/

Questions?

PENNSTATE

1855

William Enck

Systems and Internet Infrastructure Security (SIIS) Laboratory
Department of Computer Science and Engineering
The Pennsylvania State University
enck@cse.psu.edu

- Additional Team Members
  - Peter Gilbert (Duke University)
  - Byung-Gon Chun (Intel Labs, Berkeley)
  - Landon Cox (Duke University)
  - Jaeyeon Jung (Intel Labs, Seattle)
  - Patrick McDaniel (Penn State University)
  - Anmol Sheth (Intel Labs, Seattle)