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

- Dynamic taint analysis is a technique that tracks information dependencies from an origin.

- Conceptual idea:
  - Taint source
  - Taint propagation
  - Taint sink

- **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


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

---

<table>
<thead>
<tr>
<th>Op Format</th>
<th>Op Semantics</th>
<th>Taint Propagation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>const-op vA C</code></td>
<td><code>vA ← C</code></td>
<td><code>τ(vA) ← ∅</code></td>
<td>Clear <code>vA</code> taint</td>
</tr>
<tr>
<td><code>move-op vA vB</code></td>
<td><code>vA ← vB</code></td>
<td><code>τ(vA) ← τ(vB)</code></td>
<td>Set <code>vA</code> taint to <code>vB</code> taint</td>
</tr>
<tr>
<td><code>move-op-R vA</code></td>
<td><code>vA ← R</code></td>
<td><code>τ(vA) ← τ(R)</code></td>
<td>Set <code>vA</code> taint to return taint</td>
</tr>
<tr>
<td><code>return-op vA</code></td>
<td><code>R ← vA</code></td>
<td><code>τ(R) ← τ(vA)</code></td>
<td>Set return taint (∅ if void)</td>
</tr>
<tr>
<td><code>move-op-E vA</code></td>
<td><code>vA ← E</code></td>
<td><code>τ(vA) ← τ(E)</code></td>
<td>Set <code>vA</code> taint to exception taint</td>
</tr>
<tr>
<td><code>throw-op vA</code></td>
<td><code>E ← vA</code></td>
<td><code>τ(E) ← τ(vA)</code></td>
<td>Set exception taint</td>
</tr>
<tr>
<td><code>unary-op vA vB</code></td>
<td><code>vA ← ⊕vB</code></td>
<td><code>τ(vA) ← τ(vB)</code></td>
<td>Set <code>vA</code> taint to <code>vB</code> taint</td>
</tr>
<tr>
<td><code>binary-op vA vB vC</code></td>
<td><code>vA ← vB ⊗ vC</code></td>
<td><code>τ(vA) ← τ(vB) ∪ τ(vC)</code></td>
<td>Set <code>vA</code> taint to <code>vB</code> taint ∪ <code>vC</code> taint</td>
</tr>
<tr>
<td><code>binary-op vA vB C</code></td>
<td><code>vA ← vB ⊗ C</code></td>
<td><code>τ(vA) ← τ(vB)</code></td>
<td>Set <code>vA</code> taint to <code>vB</code> taint</td>
</tr>
<tr>
<td><code>aput-op vA vB vC</code></td>
<td><code>vB[vC] ← vA</code></td>
<td><code>τ(vB[vC]) ← τ(vA)</code></td>
<td>Update array <code>vB</code> taint with <code>vA</code> taint</td>
</tr>
<tr>
<td><code>aget-op vA vB vC</code></td>
<td><code>vA ← vB[vC]</code></td>
<td><code>τ(vA) ← τ(vB[vC]) ∪ τ(vC)</code></td>
<td>Set <code>vA</code> taint to array and index taint</td>
</tr>
<tr>
<td><code>spu-op vA FB</code></td>
<td><code>FB ← vA</code></td>
<td><code>τ(FB) ← τ(vA)</code></td>
<td>Set field <code>FB</code> taint to <code>vA</code> taint</td>
</tr>
<tr>
<td><code>sget-op vA FB</code></td>
<td><code>vA ← FB</code></td>
<td><code>τ(vA) ← τ(FB)</code></td>
<td>Set <code>vA</code> taint to field <code>FB</code> taint</td>
</tr>
<tr>
<td><code>iput-op vA vB FC</code></td>
<td><code>vB(FC) ← vA</code></td>
<td><code>τ(vB(FC)) ← τ(vA)</code></td>
<td>Set field <code>FC</code> taint to <code>vA</code> taint</td>
</tr>
<tr>
<td><code>iget-op vA vB FC</code></td>
<td><code>vA ← vB(FC)</code></td>
<td><code>τ(vA) ← τ(vB(FC)) ∪ τ(vB)</code></td>
<td>Set <code>vA</code> taint to field <code>FC</code> and object reference taint</td>
</tr>
</tbody>
</table>
**Data flow:** propagate source regs to destination reg

<table>
<thead>
<tr>
<th>Op Format</th>
<th>Op Semantics</th>
<th>Taint Propagation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>const-op $v_A C$</td>
<td>$v_A \leftarrow C$</td>
<td>$\tau(v_A) \leftarrow \emptyset$</td>
<td>Clear $v_A$ taint</td>
</tr>
<tr>
<td>move-op $v_A v_B$</td>
<td>$v_A \leftarrow v_B$</td>
<td>$\tau(v_A) \leftarrow \tau(v_B)$</td>
<td>Set $v_A$ taint to $v_B$ taint</td>
</tr>
<tr>
<td>move-op-R $v_A$</td>
<td>$v_A \leftarrow R$</td>
<td>$\tau(v_A) \leftarrow \tau(R)$</td>
<td>Set $v_A$ taint to return taint</td>
</tr>
<tr>
<td>return-op $v_A$</td>
<td>$R \leftarrow v_A$</td>
<td>$\tau(R) \leftarrow \tau(v_A)$</td>
<td>Set return taint ($\emptyset$ if void)</td>
</tr>
<tr>
<td>move-op-E $v_A$</td>
<td>$v_A \leftarrow E$</td>
<td>$\tau(v_A) \leftarrow \tau(E)$</td>
<td>Set $v_A$ taint to exception taint</td>
</tr>
<tr>
<td>throw-op $v_A$</td>
<td>$E \leftarrow v_A$</td>
<td>$\tau(E) \leftarrow \tau(v_A)$</td>
<td>Set exception taint</td>
</tr>
<tr>
<td>unary-op $v_A v_B$</td>
<td>$v_A \leftarrow \otimes v_B$</td>
<td>$\tau(v_A) \leftarrow \tau(v_B)$</td>
<td>Set $v_A$ taint to $v_B$ taint</td>
</tr>
<tr>
<td>binary-op $v_A v_B$</td>
<td>$v_A \leftarrow v_B \oplus v_B$</td>
<td>$\tau(v_A) \leftarrow \tau(v_B) \cup \tau(v_B)$</td>
<td>Set $v_A$ taint to $v_B$ taint</td>
</tr>
<tr>
<td>aget-op $v_A v_B v_C$</td>
<td>$v_A \leftarrow v_B[v_C]$</td>
<td>$\tau(v_A) \leftarrow \tau(v_B[\cdot]) \cup \tau(v_C)$</td>
<td></td>
</tr>
<tr>
<td>aput-op $v_A v_B v_C$</td>
<td>$v_B[v_C] \leftarrow v_A$</td>
<td>$\tau(v_B[\cdot]) \leftarrow \tau(v_B[\cdot]) \cup \tau(v_A)$</td>
<td>Update array $v_B$ taint with $v_A$ taint</td>
</tr>
<tr>
<td>get-op $v_A f_B$</td>
<td>$f_B \leftarrow v_A$</td>
<td>$\tau(f_B) \leftarrow \tau(v_A)$</td>
<td>Set field $f_B$ taint to $v_A$ taint</td>
</tr>
<tr>
<td>set-op $v_A f_B$</td>
<td>$v_A \leftarrow f_B$</td>
<td>$\tau(v_A) \leftarrow \tau(f_B)$</td>
<td>Set $v_A$ taint to field $f_B$ taint</td>
</tr>
<tr>
<td>iput-op $v_A v_B f_C$</td>
<td>$v_B(f_C) \leftarrow v_A$</td>
<td>$\tau(v_B(f_C)) \leftarrow \tau(v_A)$</td>
<td>Set field $f_C$ taint to $v_A$ taint</td>
</tr>
<tr>
<td>iget-op $v_A v_B f_C$</td>
<td>$v_A \leftarrow v_B(f_C)$</td>
<td>$\tau(v_A) \leftarrow \tau(v_B(f_C)) \cup \tau(v_B)$</td>
<td>Set $v_A$ taint to field $f_C$ and object reference taint</td>
</tr>
</tbody>
</table>
### DEX Propagation Logic

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

<table>
<thead>
<tr>
<th>Op Format</th>
<th>Op Semantics</th>
<th>Taint Propagation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>const-op $v_A C$</td>
<td>$v_A \leftarrow C$</td>
<td>$\tau(v_A) \leftarrow \emptyset$</td>
<td>Clear $v_A$ taint</td>
</tr>
<tr>
<td>move-op $v_A v_B$</td>
<td>$v_A \leftarrow v_B$</td>
<td>$\tau(v_A) \leftarrow \tau(v_B)$</td>
<td>Set $v_A$ taint to $v_B$ taint</td>
</tr>
<tr>
<td>move-op-R $v_A$</td>
<td>$v_A \leftarrow R$</td>
<td>$\tau(v_A) \leftarrow \tau(R)$</td>
<td>Set $v_A$ taint to return taint</td>
</tr>
<tr>
<td>return-op $v_A$</td>
<td>$R \leftarrow v_A$</td>
<td>$\tau(R) \leftarrow \tau(v_A)$</td>
<td>Set return taint ($\emptyset$ if void)</td>
</tr>
<tr>
<td>move-op-E $v_A$</td>
<td>$v_A \leftarrow E$</td>
<td>$\tau(v_A) \leftarrow \tau(E)$</td>
<td>Set $v_A$ taint to exception taint</td>
</tr>
<tr>
<td>throw-op $v_A$</td>
<td>$E \leftarrow v_A$</td>
<td>$\tau(E) \leftarrow \tau(v_A)$</td>
<td>Set exception taint</td>
</tr>
<tr>
<td>unary-op $v_A v_B$</td>
<td>$v_A \leftarrow \otimes v_B$</td>
<td>$\tau(v_A) \leftarrow \tau(v_B)$</td>
<td>Set $v_A$ taint to $v_B$ taint</td>
</tr>
<tr>
<td>binary-op $v_A v_B$</td>
<td>$v_A \leftarrow v_B \otimes v_B$</td>
<td>$\tau(v_A) \leftarrow \tau(v_B) \cup \tau(v_B)$</td>
<td>Set $v_A$ taint to $v_B$ taint</td>
</tr>
</tbody>
</table>

**Table 2: DEX Taint Propagation Logic**

Register variables and class fields are referenced by `←` as class instance field and arrays. These variable types have

- **RETURN AND EXCEPTION VARIABLES**
  - `valueOf()` includes the object reference taint tag in the instance
  - `static class valueOfCache {` is passed `t` at the .th USENIX Symposium on Operating Systems Design and Implementation tOSDI'21`

- `AINT` `binary-op` `move-op-E` `move-op-R` `return-op` `throw-op` `unary-op` `binary-op` `aput-op` `aget-op` `sput-op` `sget-op` `iput-op` `iget-op` `ipse-op`...
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

- 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

![Diagram showing message-level tracking and different tracking levels](image-url)
Performance

CaffeineMark 3.0 benchmark (higher is better)

- 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)

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

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

- 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=062A1CB1D476DE85B717D9195A6722A9&d%5Bcoord%5D=47.66122789000006%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

• 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/

TaintDroid running on Nexus One

* video produced by Peter Gilbert (gilbert@cs.duke.edu)
* special thanks to Gabriel Maganis (maganis@cs.ucdavis.edu) for TaintDroid UI
Questions?

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)