Evaluating Mobile Smartphone Application Security: The First Six Years

Professor Patrick McDaniel
Urbana-Champaign, IL
September 24, 2015
A cautionary tale ...
2008 View: Security and smartphones

- Smartphones: long awaited realization of mobile computing
- Usage model is very different
  - Multi-user single machine to single-user multiple machines
  - Always on, always computing social instrument
  - Enterprise: separate action from geography
- Changing Risk
  - Necessarily contains secrets (financial, personal)
  - Collects sensitive data as a matter of operation
  - Drifts seamlessly between “unknown” environments
  - Highly malleable development practices, largely unknown developers
Rethinking (host) Security

\[ \text{security} = \text{permissions} \times \text{apps} \]

\[ \text{security} \neq \text{users} \]

- **Permissions** define capabilities.
- **Application markets** deliver packaged applications from largely unknown sources.
- Users make permission decisions.
- Applications are run within middleware supported sandboxes provided by the OS.

- **Note**: App markets don’t (and can’t) provide security.
A 7-Year Span ...

- Evaluating Android Application Security …. 

2009
Permission Analysis
[CCS ’09]

2010
System Dynamic Analysis
[OSDI ’10]

2011
Static Analysis
[USENIX Sec ’11]

2012
Bytecode Retargeting
[FSE ’12]

2013
ICC Analysis
[USENIX Sec ’13]

2014
Application Dynamic Analysis
[PLDI ’14]

2015
Enhanced ICC Analysis
[ICSE ’15]

2015
Market-Scale Analysis
[in submission]
Example: Android Security

• Permissions granted to applications and *never* changed
  ‣ Permissions allow an *application* to accesses a *component, API, ..*
  ‣ Runtime decisions look for assigned permissions
    (*access is granted IFF app A assigned perm X at install*)
  ‣ Permissions levels: normal, dangerous, signature, or system

• *Example permissions*: location, phone IDs, microphone, camera, address book, SMS, application “interfaces”

What do applications ask for?

- *Kirin* certifies applications by vetting policies at install-time (relies on *runtime enforcement*)
- Obvious insight: app config and security policy is an upper bound on runtime behavior.
- Kirin is a modified application installer
  - Apps with unsafe policies are rejected

Where's the system policy?

Kirin Security Policy

• Kirin enforces security invariants at install-time
  ‣ Signatures of “malicious permission sets” approach

• Local evaluation of two manifest artifacts
  ‣ The collection of requested permissions (uses-permission)
  ‣ The types of registered Intent message listeners (i.e., sigs)

• Example:
  ‣ Do not allow an application with Location and Internet permissions and receives the “booted” event

```
restrict permission [ACCESS_FINE_LOCATION, INTERNET]
and receive [BOOT_COMPLETE]
```
Policy Evaluation

- Policy evaluation is the satisfiability of invariants
  - Invariant violations found in $O(n)$ w.r.t. policy size
- Model:
  - KSL rules are tuples: $r_i = (P_i, A_i)$
  - Configuration policy is a tuple: $c_t = (P_t, A_t)$
  - $\text{fail}(c_t, r_i)$ if $(P_t, A_t) = c_t, (P_i, A_i) = r_i, P_i \subseteq P_t \land A_i \subseteq A_t$
  - Certified if
    \[
    F_R(c_t) = \{ r_i | r_i \in R, \text{fail}(c_t, r_i) \} = 0
    \]
Studying the (early) Market

• Historical context
  ‣ September 23, 2008 – Android released
  ‣ Version 1.0, app market begins

• Evaluate 311* popular Market apps (Jan 2009)
  ‣ 5 had both dangerous configuration and functionality (1.6%)
  ‣ 5 dangerous configs, but plausible use of permissions (1.6%)

3 apps failed -- (2) An application must not have the PHONE_STATE, RECORD_AUDIO, and INTERNET permissions.

(1) An application must not have the SET_DEBUG_APP permission.
(2) An application must not have the PHONE_STATE, RECORD_AUDIO, and INTERNET permissions.
(3) An application must not have the PROCESS_OUTGOING_CALL, RECORD_AUDIO, and INTERNET permissions.
(4) An application must not have the ACCESS_FINE_LOCATION, INTERNET, and RECEIVE_BOOT_COMPLETE permissions.
(5) An application must not have the ACCESS_COARSE_LOCATION, INTERNET, and RECEIVE_BOOT_COMPLETE permissions.
(6) An application must not have the RECEIVE_SMS and WRITE_SMS permissions.
(7) An application must not have the SEND_SMS and WRITE_SMS permissions.
(8) An application must not have the INSTALL_SHORTCUT and UNINSTALL_SHORTCUT permissions.
(9) An application must not have the SET_PREFERRED_APPLICATION permission and receive Intents for the CALL action string.
Q2: What do the applications do?

- TaintDroid is performs system-wide taint tracking in the Android platform
  1. VM Layer: variable tracking throughout Dalvik VM
  2. Native Layer: patches state after native method invocation (JNI)
  3. Binder IPC Layer: extends tracking between applications
  4. Storage Layer: persistent tracking on files

Dynamic Taint Analysis

• Dynamic taint analysis is a technique that tracks information flow from an origin to output

• Conceptual idea:
  ‣ Taint source
  ‣ Taint propagation
  ‣ Taint sink

• Limitations: performance and granularity is a trade-off

\[
\begin{align*}
  c &= \text{taint\_source}() \\
  a &= b + c \\
  \text{network\_send}(a)
\end{align*}
\]
**DEX Propagation Logic**

- **Data flow**: propagate taints source reg to destination reg based on DEX (Dalvik executable) bytecode

<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>\tau(v_A) ← \emptyset</code></td>
<td>Clear <code>v_A</code> taint</td>
</tr>
<tr>
<td><code>move-op vA vB</code></td>
<td><code>vA ← vB</code></td>
<td><code>\tau(v_A) ← \tau(v_B)</code></td>
<td>Set <code>v_A</code> taint to <code>v_B</code> taint</td>
</tr>
<tr>
<td><code>move-op-R vA</code></td>
<td><code>v_A ← R</code></td>
<td><code>\tau(R) ← \tau(v_A)</code></td>
<td>Set <code>v_A</code> taint to return taint</td>
</tr>
<tr>
<td><code>return-op vA</code></td>
<td><code>R ← v_A</code></td>
<td><code>\tau(R) ← \tau(v_A)</code></td>
<td>Set return taint (\emptyset if void)</td>
</tr>
<tr>
<td><code>move-op-E vA</code></td>
<td><code>vA ← E</code></td>
<td><code>\tau(v_A) ← \tau(E)</code></td>
<td>Set <code>v_A</code> taint to exception taint</td>
</tr>
<tr>
<td><code>throw-op vA</code></td>
<td><code>E ← v_A</code></td>
<td><code>\tau(E) ← \tau(v_A)</code></td>
<td>Set exception taint</td>
</tr>
<tr>
<td><code>unary-op vA vB</code></td>
<td><code>vA ← \oplus vB</code></td>
<td><code>\tau(v_A) ← \tau(v_B)</code></td>
<td>Set <code>v_A</code> taint to <code>v_B</code> taint</td>
</tr>
<tr>
<td><code>binary-op vA vB vC</code></td>
<td><code>vA ← vB \oplus vC</code></td>
<td><code>\tau(v_A) ← \tau(v_B) \cup \tau(v_C)</code></td>
<td>Set <code>v_A</code> taint to <code>v_B</code> taint \cup <code>v_C</code> taint</td>
</tr>
<tr>
<td><code>binary-op vA vB</code></td>
<td><code>vA ← vB \otimes vB</code></td>
<td><code>\tau(v_A) ← \tau(v_B) \cup \tau(v_B)</code></td>
<td>Update <code>v_A</code> taint with <code>v_B</code> taint</td>
</tr>
<tr>
<td><code>binary-op vA vB vC</code></td>
<td><code>vA ← vB[vC]</code></td>
<td><code>\tau(v_A) ← \tau(v_B[vC]) \cup \tau(v_A)</code></td>
<td>Set <code>v_A</code> taint to <code>v_B</code> taint</td>
</tr>
<tr>
<td><code>aput-op vA vB vC</code></td>
<td><code>vB[vC] ← vA</code></td>
<td><code>\tau(v_B[vC]) ← \tau(v_A)</code></td>
<td>Update array <code>v_B</code> taint with <code>v_A</code> taint</td>
</tr>
<tr>
<td><code>aget-op vA vB vC</code></td>
<td><code>vA ← vB[vC]</code></td>
<td><code>\tau(v_A) ← \tau(v_B[vC]) \cup \tau(v_A)</code></td>
<td>Set <code>v_A</code> taint to array and index taint</td>
</tr>
<tr>
<td><code>sput-op vA fB</code></td>
<td><code>fB ← vA</code></td>
<td><code>\tau(fB) ← \tau(v_A)</code></td>
<td>Set field <code>f_B</code> taint to <code>v_A</code> taint</td>
</tr>
<tr>
<td><code>sget-op vA fB</code></td>
<td><code>vA ← fB</code></td>
<td><code>\tau(v_A) ← \tau(fB)</code></td>
<td>Set <code>v_A</code> taint to field <code>f_B</code> taint</td>
</tr>
<tr>
<td><code>iput-op vA vB fC</code></td>
<td><code>vB(fC) ← vA</code></td>
<td><code>\tau(v_B(fC)) ← \tau(v_A)</code></td>
<td>Set field <code>f_C</code> taint to <code>v_A</code> taint</td>
</tr>
<tr>
<td><code>iget-op vA vB fC</code></td>
<td><code>vA ← vB(fC)</code></td>
<td><code>\tau(v_A) ← \tau(v_B(fC)) \cup \tau(v_B)</code></td>
<td>Set <code>v_A</code> taint to <code>f_C</code> and obj. ref. taint</td>
</tr>
</tbody>
</table>
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
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
  - Survey showed 2,844 JNI methods included with OS.
  - Heuristic: return value taint gets union of input taints.
    - Leads to possibility of false positives.

*Note:* accurately measuring information flow of JNI and Android APIs is an open problem.
IPC and File Propagation

- TaintDroid uses *message level* tracking for IPC
  - Applications marshall and unmarshall individual data items
  - Modify Binder IPC libraries to bundle taint tag

- Persistent storage tracked at the *file level*
  - Single taint tag stored in the file system XATTR
  - Add XATTR support to YAFFS2 and use ext2 for SDcard
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.
Application Study (2010)

- 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

• 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=47.661227890000006%2C-122.31589477&...

• 7 applications sent device (IMEI) and 2 apps sent phone info (Ph. #, IMSI, ICC-ID) to a remote server without informing the user.
Aside: Dalvik EXecutables

• Android applications written in Java, compiled to Java bytecode, and translated into DEX bytecode (Dalvik VM)

• We want to work with Java (bc), not DEX bytecode
  ‣ There are a lot of existing program analysis tools for Java
  ‣ We want to see what the developer was doing (i.e., confirmation)

• Non-trivial to retarget back to Java: register vs. stack architecture, constant pools, ambiguous scalar types, null references, etc.
Getting back to the Source

• The ded (later dare) decompiler
  ‣ Refers to both the entire process and .dex ⇒ .class retargeting tool

• ded/dare recovers logic

• from application package
  ‣ Retargeting: type inference, instruction translation, etc
  ‣ Optimization: use Soot to optimize Java bytecode
  ‣ Decompilation/IR: standard Java decompilation (Soot), or translate to TyDe IR (typed dex in DARE)

What can the applications do?

- **Static analysis**: look at the possible paths and interaction of data
  - Very, very hard (often undecidable), but community has learned that we can do a lot with small analyses.

- Step 1: decompiler for Android applications (ded)

- Step 2: static source code analysis for both *dangerous functionality* and *vulnerabilities*
  - What data could be exfiltrated from the application?
  - Are developers safely using interfaces?
Studying Application Security

- Decomplied top 1,100 apps from Android market: over \textit{21 MLOC}

- Queried for security properties using \textit{program analysis}, followed by \textit{manual inspection} to understand purpose

- Used several types of analysis to design security properties specific to Android using the Fortify SCA framework

\begin{tabular}{|l|l|}
\hline
\textbf{Misuse of Phone Identifiers} & Data flow analysis \\
\textbf{Exposure of Physical Location} & Data flow analysis \\
\textbf{Abuse of Telephony Services} & Semantic analysis \\
\textbf{Eavesdropping on Video} & Control flow analysis \\
\textbf{Eavesdropping on Audio} & Structural analysis (+CG) \\
\textbf{Botnet Characteristics (Sockets)} & Structural analysis \\
\textbf{Havesting Installed Applications} & Structural analysis \\
\hline
\end{tabular}

\begin{tabular}{|l|l|}
\hline
\textbf{Analysis for Dangerous Behavior} & \\
\textbf{Analysis for Vulnerabilities} & \\
\hline
\textbf{Leaking Information to Logs} & Data flow analysis \\
\textbf{Leaking Information to IPC} & Control flow analysis \\
\textbf{Unprotected Broadcast Receivers} & Control flow analysis \\
\textbf{Intent Injection Vulnerabilities} & Control flow analysis \\
\textbf{Delegation Vulnerabilities} & Control flow analysis \\
\textbf{Null Checks on IPC Input} & Control flow analysis \\
\textbf{Password Management\textsuperscript{*}} & Data flow analysis \\
\textbf{Cryptography Misuse\textsuperscript{*}} & Structural analysis \\
\textbf{Injection Vulnerabilities\textsuperscript{*}} & Data flow analysis \\
\hline
\end{tabular}

* included with analysis framework

Also studied inclusion of advertisement and analytics libraries and associated properties.
Phone Identifiers

- Analysis pin-pointed 33 apps leaking Phone IDs

```java
public String toUrlFormatedString()
{
    StringBuilder $r4;
    if (mURLFormatedParameters == null)
    {
        $r4 = new StringBuilder();
        $r4.append((new StringBuilder("&uuid=")).append(URLEncoder.encode(mUuid)).toString());
        $r4.append((new StringBuilder("&device=")).append(URLEncoder.encode(mModel)).toString());
        $r4.append((new StringBuilder("&platform=")).append(URLEncoder.encode(mOSVersion)).toString());
        $r4.append((new StringBuilder("&ver=")).append(mAppVersion).toString());
        $r4.append((new StringBuilder("&app=")).append(this.getAppName()).toString());
        $r4.append("&returnfmt=json");
        mURLFormatedParameters = $r4.toString();
    }

    return mURLFormatedParameters;
}
```
com.froogloid.kring.google.zxing.client.android - Activity_Router.java (Main Activity)

```java
public void onCreate(Bundle r1) {
    ...
    IMEI = ((TelephonyManager) this.getSystemService("phone")).getDeviceId();
    retailerLookupCmd = (new StringBuilder(String.valueOf(constants.server))).append("identifier=").append(EncodeURL.KREncodeURL(IMEI)).append("&command=retailerlookup&retailername=").toString();
    ...
}
```

http://kror.keyringapp.com/service.php

com.Qunar - net/NetworkTask.java

```java
public void run() {
    ...
    r24 = (TelephonyManager) r21.getSystemService("phone");
    url = (new StringBuilder(String.valueOf(url))).append("&vid=60001001&pid=10010&cid=C1000&uid=").append(r24.getDeviceId()).append("&gid=").append(QConfiguration.mGid).append("&msg=").append(QC onfiguration.getInstance().mPCStat.toMsgString()).toString();
    ...
}
```

http://client.qunar.com:80/QSearch
com.statefarm.pocketagent - activity/LogInActivity$1.java (Button callback)

```java
public void onClick(View r1) {
    ...
    r7 = Host.getDeviceId(this$0.getApplicationContext());
    LogInActivity.access$1(this$0).setUniqueDeviceID(r7);
    this$0.loginTask = new LogInActivity$LoginTask(this$0, null);
    this$0.showProgressDialog(r2, 2131361798, this$0.loginTask);
    r57 = this$0.loginTask;
    r58 = new LoginTO[1];
    r58[0] = LogInActivity.access$1(this$0);
    r57.execute(r58);
    ...
}
```

How would you feel about a PII to phone database?
public static String getDeviceId(Context r0) {
    String r1;
    r1 = "";
    label_19:
    {
        if (deviceId != null)
        {
            if (r1.equals(deviceId) == false)
            {
                break label_19;
            }
        }
        if (r0.checkCallingOrSelfPermission("android.permission.READ_PHONE_STATE") == 0)
        {
            deviceId = ((TelephonyManager) r0.getSystemService("phone")).getSubscriberId();
        }
    } //end label_19:
    ...
}
Ad/Analytics Libraries

- 51% of the apps included an ad or analytics library (many also had custom functionality)
- A few libraries were used most frequently
- Use of phone identifiers and location sometimes configurable by developer

### Library Path, # Apps, Obtains

<table>
<thead>
<tr>
<th>Library Path</th>
<th># Apps</th>
<th>Obtains</th>
</tr>
</thead>
<tbody>
<tr>
<td>com/admob/android/ads</td>
<td>320</td>
<td>L</td>
</tr>
<tr>
<td>com/google/ads</td>
<td>206</td>
<td>-</td>
</tr>
<tr>
<td>com/flurry/android</td>
<td>98</td>
<td>-</td>
</tr>
<tr>
<td>com/qwapi/adclient/android</td>
<td>74</td>
<td>L, P, E</td>
</tr>
<tr>
<td>com/google/android/apps/analytics</td>
<td>67</td>
<td>-</td>
</tr>
<tr>
<td>com/adwhirl</td>
<td>60</td>
<td>L</td>
</tr>
<tr>
<td>com/mobclix/android/sdk</td>
<td>58</td>
<td>L, E</td>
</tr>
<tr>
<td>com/mellennialmedia/android</td>
<td>52</td>
<td>-</td>
</tr>
<tr>
<td>com/zestadz/android</td>
<td>10</td>
<td>-</td>
</tr>
<tr>
<td>com/admarvel/android/ads</td>
<td>8</td>
<td>-</td>
</tr>
<tr>
<td>com/estsoft/adlocal</td>
<td>8</td>
<td>L</td>
</tr>
<tr>
<td>com/adfonic/android</td>
<td>5</td>
<td>-</td>
</tr>
<tr>
<td>com/vdroid/ads</td>
<td>5</td>
<td>L, E</td>
</tr>
<tr>
<td>com/greystripe/android/sdk</td>
<td>4</td>
<td>E</td>
</tr>
<tr>
<td>com/medialets</td>
<td>4</td>
<td>L</td>
</tr>
<tr>
<td>com/wooboo/adlib_android</td>
<td>4</td>
<td>L, P, I</td>
</tr>
<tr>
<td>com/adserver/adview</td>
<td>3</td>
<td>L</td>
</tr>
<tr>
<td>com/tapjoy</td>
<td>3</td>
<td>-</td>
</tr>
<tr>
<td>com/inmobi/androidsdk</td>
<td>2</td>
<td>E</td>
</tr>
<tr>
<td>com/apegroup/ad</td>
<td>1</td>
<td>-</td>
</tr>
<tr>
<td>com/casee/adsdk</td>
<td>1</td>
<td>S</td>
</tr>
<tr>
<td>com/webtrents/mobile</td>
<td>1</td>
<td>L, E, S, I</td>
</tr>
<tr>
<td><strong>Total Unique Apps</strong></td>
<td>561</td>
<td></td>
</tr>
</tbody>
</table>

L = Location; P = Ph#; E = IMEI; S = IMSI; I = ICC-ID
Intent Vulnerabilities

- Similar analysis rules as independently verified by Chin et al. [Mobisys 2011]
- **Leaking information to IPC** - unprotected intent broadcasts are common, occasionally contain sensitive info
- **Unprotected broadcast receivers** - a few apps receive custom action strings w/out protection (lots of “protected bcasts”)
- **Intent injection attacks** - 16 apps had potential vulnerabilities
- **Delegating control** - pending intents are tricky to analyze (notification, alarm, and widget APIs) --- no vulns found
- **Null checks on IPC input** - 3925 potential null dereferences in 591 apps (53%) --- most were in activity components
Data Flow (revisited)

• Application analysis is more challenging because of application execution “life-cycle”
  ‣ E.g., component asynchrony, multiple entry points, system events, callbacks …

• FlowDroid is a static taint analysis system that tracks data flow from sources to sinks
  ‣ Approach: identify all entry points construct a dummy main, perform analysis

• Analysis: 93% recall and 86% precision
  ‣ DroidBench (39 hand crafted applications)

• Market or enterprise level analysis
  ‣ Getting back to the certification model of Kirin

How do applications work on concert?

• *Intents* are used to pass information between apps

• IPC (*intra-* ) and component (*inter-application*) data flows

• **ICC Analysis**: location of ICC, and data (types, attributes)
  ‣ Soundness: all Intra-Component-Communication (ICC) identified
  ‣ Precision: reduce number of false positives
  ‣ Enable security analysis of ensemble of applications
    • Data flows between components within application
    • Exported flows/interfaces are used by other applications

---

Analysis Results

- Epicc builds a *model* of ICC
  - Reduce to an *Interprocedural Distributive Environment (IDE)* problem and extract possible Intent values (specifications)
  - Experiment: attempt to recover Intent use in 1200 applications (850 most popular, 350 random applications),
    - Runtime: average 113/seconds per application
  - Entry/exit point analysis
    - All attributes known in about 93% of ICC specifications
    - 56,106 exits points
      - 90% were found to have fixed Intent specification
      - 45% have key-value data
    - 29,154 entry points
      - About 95% were found to have single Intent Filter specification
      - 8,566 exported components, *5% protected by permissions*
ICC Analysis version 2.0

- IC3: Inter-Component Communication Analysis in Android with COAL
  - More sophisticated two-phase string analysis using flow graph of constraints on string operations
  - Added deeper URI analysis
  - Experiment: analyze ICC for 460 apps using IC3 and Epicc

<table>
<thead>
<tr>
<th></th>
<th>EPICC</th>
<th>IC3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intents/Filters</td>
<td>69%</td>
<td>86%</td>
</tr>
<tr>
<td>URIs</td>
<td>34%</td>
<td>72%</td>
</tr>
<tr>
<td>Total</td>
<td>66%</td>
<td>85%</td>
</tr>
</tbody>
</table>

Identified (possible) ICC Flows
- Epicc: 120,817
- IC3: 26,872

Ongoing: Scaling up analysis ...

- Static analysis
  - Epicc/EC2: find all Intent values at message-passing program points
  - Small static analysis imprecisions cause explosion in number of links at large scale
  - 600 apps -> 2 million links!

- Two challenges
  - Intent resolution
  - Intent flow ranking
1: Intent Resolution

• Intent resolution – identifying the flows between apps
  ‣ Compute inter-component links in scalable manner
  ‣ Take into account field value regular expressions
  ‣ Our algorithm is based on intersecting sets of Filters that verify several Intent resolution tests
    • Exploits fast matching of constant intent values
  ‣ Runs in time $O(e \cdot |I|)$, where $e$ is small constant

• Experiment
  ‣ Match 10,928 applications
  ‣ Runtime: 8,434 seconds (140 min)
2: Intent flow ranking

• Intent flow ranking – determining estimated likelihood of flows being “real” by comparing against known flows

• Idea: Intents are highly predictable
  ‣ For example, displaying a map is done by sending Intent with VIEW action and geo scheme (common to applications)
  ‣ Explicit Intents almost always target components within the same application, but often identified as being inter-application

• Approach
  ‣ Estimate the probability of having a given Intent field combination, given the Intents that are known, i.e., to simplify

\[ P(\text{flow}) = \% \text{ known Intent matching specifications matching Intent filter} \]

• Intuition: how similar is potential flow to known flows
Preliminary Results

- 10,928 applications, 489,099,606 potential ICC flows
- 111,254 components, 58,480 Intent filters
- 452,984 Intent values (47% explicit, 53% implicit)

Key Results
- 97.3% links $Pr() < 0.1$
- 75% explicit links are tagged as inter-application
- 99.6% of Implicit links are inter-application

Take away: vastly reduces the number of links
Conclusions

• The security community has been analyzing mobile applications for almost a decade now..
  ‣ Research is driven by deep and deeper questions
    • Analysis techniques are getting more sophisticated …
  ‣ Volumes of data are getting larger …
  ‣ Adversarial behavior getting subtler and more costly …
  ‣ Industrial and academic cooperation is very strong …
  ‣ Future is bright for research!
Contact

Patrick McDaniel
mcdaniel@cse.psu.edu
http://siis.cse.psu.edu/
http://patrickmcdaniel.org/