Interactively Verifying Absence of Explicit Information Flows in Android Apps

Osbert Bastani, Saswat Anand, and Alex Aiken
Stanford University
OOPSLA 2015
Problem

• Google Play Store
  • > 1 million apps on the store

• Lots of malware submitted
  • Information leaks
  • SMS Fraud
  • Ransomware
Information Flow Analysis

• Finding Android malware using source to sink flows

  Information leak: location flows to Internet
  SMS Fraud: phone # used in SMS send
  Ransomware: network packets encrypt files
Standard Audits

- **No** static information flow
  - Too many false positives

- Light-weight static analysis
  - Dynamic code loading
  - Calls to undocumented APIs

- Dynamic analysis
  - Information flows
Standard Audits

Android App → Light Static + Dynamic Analysis → FNs → Google Play
Dead Code

• Dead code can cause false positive information flows
  • Global property (e.g., method with no caller)

• Examples
  • Leaks in 3rd party libraries
  • Conservative assumptions about potential callbacks
Key Issue

• Hard to understand someone else’s code
  • No source code!
  • Obfuscation

• Can we shift work to developer?
Developer Queries

Program

static analysis
• dead code
• points-to
• information flow

sound, imprecise results
Developer Queries

Program

static analysis

- dead code
- points-to
- information flow

sound, imprecise results

potential
dead code
Developer Queries

Program

→

static analysis
• dead code
• points-to
• information flow

→

sound, imprecise results

yes/no

potential dead code
Developer Queries

Program

static analysis
• dead code
• points-to
• information flow

sound, precise results

yes/no

potential
dead code
Problem

• Developer may be adversarial

• Solution: Enforce response
Developer Queries

Program

static analysis

- dead code
- points-to
- information flow

sound, imprecise results

potential dead code
Developer Queries

Program

static analysis

- dead code
- points-to
- information flow

sound, imprecise results

yes 

potential dead code
Developer Queries

Program

→ static analysis
  • dead code
  • points-to
  • information flow

→ sound, precise results

potential dead code

yes
Developer Queries

Program

static analysis
- dead code
- points-to
- information flow

sound, precise results

potential dead code

delete confirmed dead code

Program
Developer Queries

Program

→

static analysis

• dead code
• points-to
• information flow

→

sound, precise results

potential
dead code
Developer Queries

Program

static analysis
  • dead code
  • points-to
  • information flow

sound, precise results

potential dead code

no
Developer Queries

Program

 static analysis
• dead code
• points-to
• information flow

 sound, precise results

![Diagram showing test case and potential dead code](image-url)
Developer Queries

Program

\[ \text{static analysis} \]
- dead code
- points-to
- information flow

sound, precise results

no queries

filter false positives

\[
\text{test case} + \text{no} \rightarrow \text{potential dead code}
\]
Developer Queries: Cuts

• **Cut**
  • Code that is (potentially) dead
  • Removing the code breaks (potential) information flows

• **Valid cut**: Developer confirms that the cut is dead
Developer Queries: Cuts
Developer Queries: Cuts

Android Framework

- getLatitude()
- sendHTTP(…)

App

Potential Dead Code
Developer Queries: Cuts

Android Framework

getLatitude()  sendHTTP(…)

not a cut

App

Potential Dead Code
Developer Queries: Cuts

Android Framework

App

Potential Dead Code
Developer Queries: Cuts

Android Framework

- `getLatitude()`
- `sendHTTP(...)`

App

Potential Dead Code

min cut
Developer Queries: Cuts

Android Framework

- `getLatitude()`
- `sendHTTP(...)`

App

Potential Dead Code

Invalid!

min cut
Developer Queries: Cuts

Android Framework

App

Potential Dead Code

getLatitude()

sendHTTP(…)

Developer Queries: Cuts

Android Framework

- getLatitude()
- sendHTTP(...)

min cut

App

Potential Dead Code
Developer Queries: Cuts

Android Framework

- getLatitude()
- sendHTTP(…)

App

Potential Dead Code

Valid!

min cut
Developer Queries: Cuts

Android Framework

- getLatitude()
- sendHTTP(…)

App

Potential Dead Code
Developer Queries

• Developer query
  • List of statements in cut

• Developer response
  • Test case that executes reachable statements in the cut

• Why tests?
  • Verifiable
  • Developers routinely write tests
  • Can seed with dynamic analysis
  • Aid auditor
Interactive Verification

• **Step 1:** Sound static analysis
• **Step 2:** Find cut and query developer
• **Step 3:** Update potential dead code and repeat
• **Step 4:** *Delete* valid cut
Our Audits

Android App

Light Static + Dynamic Analysis

Cuts

no cut

Google Play
Finding Valid Cuts
Finding Valid Cuts

1. String **lat** = getLatitude();
2. Runnable **runMalice** =
3.     new Runnable() {
4.         **void** run() { sendHTTP(**lat**); }
5.     };
6. Runnable **runBenign** =
7.     new Runnable() {
8.         **void** run() {}
9.     };
10. **runBenign**.run();
Finding Valid Cuts

1. String `lat` = `getLatitude()`;
2. Runnable `runMalice` =
   
   ```java
   new Runnable() {
       void run() { sendHTTP(lat); }
   }
   ```
3. `runMalice.run();`
4. Runnable `runBenign` =
   ```java
   new Runnable() {
       void run() {}  
   }
   ```
5. `runBenign.run();`

**Diagram:**
- **LOCATION**
  - `getLatitude().return`
  - `sendHTTP.arg`
- **INTERNET**
Finding Valid Cuts

1. String \texttt{lat} = \texttt{getLatitude}();
2. Runnable \texttt{runMalice} =
3. \hspace{1em} \texttt{new} Runnable() {
4. \hspace{2em} \texttt{void} \texttt{run}() { \texttt{sendHTTP(lat);} }
5. \hspace{1em} }
6. Runnable \texttt{runBenign} =
7. \hspace{1em} \texttt{new} Runnable() {
8. \hspace{2em} \texttt{void} \texttt{run}() {}
9. \hspace{1em} }
10. \texttt{runBenign.run();}
Finding Valid Cuts

1. String `lat = getLatitude();`
2. Runnable `runMalice =`
3. `new Runnable() {`
4. `void run() { sendHTTP(lat); }`
5. `};`
6. Runnable `runBenign =`
7. `new Runnable() {`
8. `void run() {}`
9. `};`
10. `runBenign.run();`
Finding Valid Cuts

1. String lat = getLatitude();
2. Runnable runMalice =
3.    new Runnable() {
4.        void run() { sendHTTP(lat); }
5.    };
6. Runnable runBenign =
7.    new Runnable() {
8.        void run() {}
9.    };
10. runBenign.run();
Finding Valid Cuts

1. String `lat = getLatitude();`
2. Runnable `runMalice =`
   3.    `new` Runnable() {
        4.      `void` run() { sendHTTP(`lat`); }
        5.    };
6. Runnable `runBenign =`
   7.    `new` Runnable() {
        8.      `void` run() {}
        9.    };
10. `runBenign.run();`
Finding Valid Cuts

1. String \texttt{lat} = \texttt{getLatitude}();
2. Runnable \texttt{runMalice} =
3. \hspace{1em} \texttt{new} Runnable() {
4. \hspace{2em} \texttt{void} run() { \texttt{sendHTTP(lat);} }
5. \hspace{1em} };
6. Runnable \texttt{runBenign} =
7. \hspace{1em} \texttt{new} Runnable() {
8. \hspace{2em} \texttt{void} run() {} }
9. \hspace{1em} };
10. \texttt{runBenign.run();}
Finding Valid Cuts

1. String \texttt{lat} = getLatitude();
2. Runnable \texttt{runMalice} =
3. \hspace{1em} \texttt{new} Runnable() {
4. \hspace{2em} \texttt{void} run() { sendHTTP(\texttt{lat}); }
5. \hspace{1em} }
6. Runnable \texttt{runBenign} =
7. \hspace{1em} \texttt{new} Runnable() {
8. \hspace{2em} \texttt{void} run() {}
9. \hspace{1em} }
10. \texttt{runBenign.run();} \hspace{1em} \text{potentially dead}
Finding Valid Cuts

1. String \texttt{lat} = \texttt{getLatitude}();
2. Runnable \texttt{runMalice} =
3. \hspace{1em} \texttt{new} Runnable() {
4. \hspace{2em} \texttt{void} run() { \texttt{sendHTTP(lat)}; }
5. \hspace{1em} }
6. Runnable \texttt{runBenign} =
7. \hspace{1em} \texttt{new} Runnable() {
8. \hspace{2em} \texttt{void} run() {}
9. \hspace{1em} }
10. \texttt{runBenign}.run(); \textbf{potentially dead}

\begin{tikzpicture}[node distance=1.5cm]
    \node (lat) {getLatitude.return};
    \node (lat1) [below=of lat] {\texttt{lat}};
    \node (http1) [below=of lat1] {\texttt{sendHTTP.arg}};
    \node (http2) [below=of http1] {INTERNET};
    \node (assign1) [left=of lat1] {Assign \texttt{1}};
    \node (assign2) [left=of http1] {Assign \texttt{1}};
    \node (sink) [below=of http2] {S\texttt{rcSink}};
    \node (source) [above=of lat] {LOCATION \texttt{\infty}};
    \draw[->] (lat) -- (lat1) node[above] {\texttt{getLatitude.return}};
    \draw[->] (lat1) -- (http1) node[above] {\texttt{sendHTTP.arg}};
    \draw[->] (http1) -- (http2) node[above] {INTERNET};
    \draw[->] (lat1) -- (assign1) node[above] {Assign \texttt{1}};
    \draw[->] (http1) -- (assign2) node[above] {Assign \texttt{1}};
    \draw[->] (assign1) -- (sink) node[above] {S\texttt{rcSink}};
    \draw[->] (assign2) -- (sink) node[above] {S\texttt{rcSink}};
\end{tikzpicture}
Finding Valid Cuts

1. String \textit{lat} = getLatitude();
2. Runnable \textit{runMalice} =
3. \hspace{1em} new Runnable() {
4. \hspace{2em} void run() { sendHTTP(\textit{lat}); }
5. \hspace{1em} }
6. Runnable \textit{runBenign} =
7. \hspace{1em} new Runnable() {
8. \hspace{2em} void run() {} 
9. \hspace{1em} }
10. \textit{runBenign}.run();
Finding Valid Cuts

1. String lat = getLatitude();
2. Runnable runMalice =
3.   new Runnable() {
4.     void run() { sendHTTP(lat); }
5.   };
6. Runnable runBenign =
7.   new Runnable() {
8.     void run() {}
9.   };
10. runBenign.run();
Finding Valid Cuts

1. String lat = getLatitude();
2. Runnable runMalice =
3.     new Runnable()
4.     { void run() { sendHTTP(lat); } }
5. }
6. Runnable runBenign =
7.     new Runnable()
8.     { void run() {}
9.     }
10. runBenign.run();
Finding Valid Cuts

1. String **lat** = getLatitude();
2. Runnable **runMalice** =
3.   new Runnable() {
4.     **void** run() { sendHTTP(**lat**); }
5.   };
6. Runnable **runBenign** =
7.   new Runnable() {
8.     **void** run() {}
9.   };
10. **runBenign**.run();
Finding Valid Cuts

1. String \texttt{lat} = \texttt{getLatitude}();
2. Runnable \texttt{runMalice} =
3. \hspace{1em} \texttt{new} Runnable() {
4. \hspace{2.5em} \texttt{void} run() { \texttt{sendHTTP(lat);} }
5. \hspace{1em} }
6. Runnable \texttt{runBenign} =
7. \hspace{1em} \texttt{new} Runnable() {
8. \hspace{2.5em} \texttt{void} run() {}
9. \hspace{1em} }
10. \texttt{runBenign}.run();
Finding Valid Cuts

1. String lat = getLatitude();
2. Runnable runMalice =
3.     new Runnable() {
4.         void run() { sendHTTP(lat); }
5.     };
6. Runnable runBenign =
7.     new Runnable() {
8.         void run() {} 
9.     };
10. runBenign.run();

Valid!
Finding Valid Cuts

1. String `lat` = getLatitude();
2. Runnable `runMalice` =
3.   `new` Runnable() {
4.     `void` run() { `throw` `new` Error(); `sendHTTP`(`lat`); }
5.   };
6. Runnable `runBenign` =
7.   `new` Runnable() {
8.     `void` run() {}
9.   };
10. `runBenign`.run();
Multiple Cuts
Multiple Cuts

• What if a developer mistakenly answers “yes”?
  • Might delete important code!
Multiple Cuts

• What if a developer mistakenly answers “yes”?  
  • Might delete important code!

• Solution: **Multiple independent** cuts  
  • Developer only needs to be right once!
Multiple Cuts

1. String \texttt{lat} = \texttt{getLatitude}();
2. Runnable \texttt{runMalice} = 
3. \hspace{1em} \texttt{new} Runnable() { 
4. \hspace{2em} \texttt{void} run() \{ \texttt{sendHTTP(lat);} \} 
5. \hspace{1em} };
6. Runnable \texttt{runBenign} = 
7. \hspace{1em} \texttt{new} Runnable() { 
8. \hspace{2em} \texttt{void} run() \{ \} 
9. \hspace{1em} };
10. \texttt{runBenign}.run();
Multiple Cuts

1. String **lat** = getLatitude();
2. Runnable **runMalice** =
3.    new Runnable() {
4.        void run() { sendHTTP(**lat**); }
5.    };
6. Runnable **runBenign** =
7.    new Runnable() {
8.        void run() {} 
9.    };
10. **runBenign**.run();
Multiple Cuts

1. String \texttt{lat} = \texttt{getLatitude}();
2. Runnable \texttt{runMalice} =
3. \texttt{new} Runnable() {
4. \hspace{1em} \texttt{void} run() \{ \texttt{sendHTTP} (\texttt{lat}); \}
5. \hspace{1em}};
6. Runnable \texttt{runBenign} =
7. \texttt{new} Runnable() {
8. \hspace{1em} \texttt{void} run() {} 
9. \hspace{1em}};
10. \texttt{runBenign}.run();
Multiple Cuts

1. String lat = getLatitude();
2. Runnable runMalice =
3.     new Runnable() {
4.         void run() { sendHTTP(lat); }
5.     };
6. Runnable runBenign =
7.     new Runnable() {
8.         void run() {}
9.     };
10. runBenign.run();
Multiple Cuts

1. String `lat = getLatitude();`
2. Runnable `runMalice =`
3.     new Runnable() {
4.         void run() { sendHTTP(lat); }
5.     }
6. Runnable `runBenign =`
7.     new Runnable() {
8.         void run() {}
9.     }
10. `runBenign.run();

Valid!
Multiple Cuts

• Program correct if *any* cut is valid
• Terminate only if *every* cut is reached
Experiments

• Ran tool on corpus of 78 Android apps
• Experiment 1: Recorded cut sizes
• Experiment 2: Interactively verified cuts
Cut sizes

<table>
<thead>
<tr>
<th>#Statements</th>
</tr>
</thead>
<tbody>
<tr>
<td>14</td>
</tr>
<tr>
<td>12</td>
</tr>
<tr>
<td>10</td>
</tr>
<tr>
<td>8</td>
</tr>
<tr>
<td>6</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>0</td>
</tr>
</tbody>
</table>

<5 kLOC (16 Apps)  5-50 kLOC (26 Apps)  >50 kLOC (36 Apps)

Cut 1  Cut 2

All (78 Apps)
Cut sizes

- 0 cases
- 2 cases
- 4 cases
- 6 cases
- 8 cases
- 10 cases
- 12 cases
- 14 cases

Cut sizes for different LOC ranges:
- <5 kLOC (16 Apps): 0
- 5-50 kLOC (26 Apps): 2
- >50 kLOC (36 Apps): 14
- All (78 Apps): 0

#Statements

Red bars represent:
- Cut 1
- Cut 2

4-cut model
Cut sizes

<table>
<thead>
<tr>
<th>Cut sizes</th>
<th>&lt;5 kLOC (16 Apps)</th>
<th>5-50 kLOC (26 Apps)</th>
<th>&gt;50 kLOC (36 Apps)</th>
<th>All (78 Apps)</th>
</tr>
</thead>
<tbody>
<tr>
<td>#Statements</td>
<td>0</td>
<td>4</td>
<td>14</td>
<td>6</td>
</tr>
</tbody>
</table>

- Cut 1
- Cut 2

Legend:
- Red: Cut 1
- Black: Cut 2
Interactive Verification

# Unverified Apps Remaining

0 1 2 3 4 5 6 7 8 9 10 11 12
Interactive Verification
Interactive Verification

# Unverified Apps

Remaining

0 1 2 3 4 5 6 7 8 9 10 11 12

Input

Gen. cuts
Interactive Verification

The diagram illustrates the process of interactive verification with a bar chart showing the number of unverified apps remaining. The chart is divided into two cuts, Cut 1 and Cut 2, with Cut 1 having a higher number of remaining unverified apps compared to Cut 2. The process involves input, generation of cuts, and querying the developer for the first iteration.
Interactive Verification

![Diagram showing the process of interactive verification.](Diagram)

- **Input**
- **Gen. cuts**
- **Query developer (Iteration 1)**
- **Gen. cuts**

- **# Unverified Apps Remaining**
- **Cut 1**
- **Cut 2**

- Numbers: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12
Interactive Verification

- Input
- Gen. cuts
- Query developer (Iteration 1)
- Gen. cuts
- Query developer (Iteration 2)

# Unverified Apps Remaining

- Cut 1
- Cut 2

- Gen. cuts
- Query developer (Iteration 1)
- Gen. cuts
- Query developer (Iteration 2)
Interactive Verification

# Unverified Apps

- Remaining

Cut 1
Cut 2
Cut 1
Cut 2

Gen. cuts
Query developer (Iteration 1)
Gen. cuts
Query developer (Iteration 2)

Input
Gen. cuts
Query developer (Iteration 1)
Gen. cuts
Query developer (Iteration 2)
Output

Output
Summary

• Remove dead code with adversarial developer
  • **Step 1:** Sound static analysis
  • **Step 2:** Find cut and query developer
  • **Step 3:** Update knowledge and repeat
  • **Step 4:** *Delete* valid cut

• Experiments
  • Discharged 11 out of 12 false positives
  • Only 2 iterations needed
Conclusions

• Manual labor in “automatic” static analysis
  • Filter false positives

• A little interaction goes a long way
  • Discharged 11 out of 12 false positives due to dead code
Future Work

• Other sources of false positives
  • Reflective method calls
  • Implicit flows

• More complex security policies
References


• M. D. Ernst et al. Collaborative verification of information flow for a high-assurance app store. In CCS, 2014.


Thanks!