Effective Real-Time Android Application Auditing

Mingyuan Xia, Lu Gong, Yuanhao Lyu, Zhengwei Qi, Xue Liu

McGill University, Shanghai Jiao Tong University
App Permissions

App permissions

Facebook needs access to additional permissions (marked as NEW):

Your messages

NEW: Read your text messages (SMS or MMS)

Your personal information

NEW: Add or modify calendar events and send emails to guests without host's knowledge, read calendar events plus confidential information, read your own contact card

Network communication

NEW: Connect and disconnect from Wi-Fi

Full network access
Data Leak?

App permissions

Facebook needs access to additional permissions (marked as NEW):

Your messages
NEW: Read your text messages (SMS or MMS)

Your personal information
NEW: Add or modify calendar events and send emails to guests without host's knowledge, read calendar events plus confidential information, read your own contact card

Network communication
NEW: Connect and disconnect from Wi-Fi

Full network access
You Should!
Source and Sink

getPhoneNumber()
(source API)

“http://...&phone=5143980000”

URL.openConnection(...)
(sink API)
App Auditing

### Auditing Result

<table>
<thead>
<tr>
<th>Item</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source</td>
<td>Phone number</td>
</tr>
<tr>
<td>Sink</td>
<td>Network</td>
</tr>
<tr>
<td>Form</td>
<td>Unencrypted HTTP GET</td>
</tr>
<tr>
<td>Domain name</td>
<td>http:// ...</td>
</tr>
<tr>
<td>Component</td>
<td>com.aa.bb</td>
</tr>
</tbody>
</table>
Use Cases

App developer

Problematic libraries

Problematic developers

Problematic app markets

App market

Mobile user

AppAudit

AppAudit

AppAudit

AppAudit
Existing Tools

- Dynamic analysis
e.g. TaintDroid [OSDI’10]
Existing Tools

• Dynamic analysis
  e.g. TaintDroid [OSDI’10]
  **Limitations:** dependent on user inputs and low coverage
Existing Tools

- Dynamic analysis
e.g. TaintDroid [OSDI’10]
**Limitations:** dependent on user inputs and low coverage

- Static analysis
e.g. AppIntent [CCS’13]
FlowDroid [PLDI’14]
Existing Tools

• Dynamic analysis
  e.g. TaintDroid [OSDI’10]
  **Limitations:** dependent on user inputs and low coverage

• Static analysis
  e.g. AppIntent[CCS’13]
  FlowDroid [PLDI’14]
  **Limitations:** false positives, time-consuming
Static Analysis Meets Real Apps

• Today’s real apps
  – 10,000 ~ 100,000 functions
  – Millions of instructions
Static Analysis Meets Real Apps

• Today’s real apps
  – 10,000 ~ 100,000 functions
  – Millions of instructions

• Today’s static analysis
  (whole-program analysis)
  – Time: minutes ~ hours
  – Memory: 4GB ~ 32GB
AppAudit

• Problems:
  – Static analysis is slow
  – Dynamic analysis does not run automatically
AppAudit

• Problems:
  – Static analysis is slow
  – Dynamic analysis does not run automatically

• AppAudit design: A synergy of two analyses
  – Make static analysis faster with some false positives
  – Automates dynamic analysis, prune false positives
AppAudit

• Problems:
  – Static analysis is slow
  – Dynamic analysis does not run automatically

• AppAudit design: A synergy of two analyses
  – Make static analysis faster with some false positives
  – Automates dynamic analysis, prune false positives

Approximated execution
AppAudit

Dalvik bytecode

Coarse-grained static analysis

Suspicious functions
AppAudit

Dalvik bytecode → Coarse-grained static analysis → Leaking functions + false positives
AppAudit

Leaking functions

Dalvik bytecode

Coarse-grained static analysis

Leaking functions + false positives

Approximated execution
AppAudit

Leaking functions

Dalvik bytecode → Coarse-grained static analysis

Leaking functions + false positives

Approximated execution

dynamic parallel
Lightweight Static Analysis

• Goal: find suspicious functions
  – Find functions that can reach both a source and sink API

- getPhoneNumer()
- openConnection()
Lightweight Static Analysis

• Goal: find suspicious functions
  – Find functions that can reach both a source and sink API

• Over-estimating call graph
  – Efficient
  – Some false positives
Dynamic Analysis

• Approximated execution
  – Automatically and approximately execute a suspicious function to confirm leaks
  – Mimic real execution
Dynamic Analysis

• Approximated execution
  – Automatically and approximately execute a suspicious function to confirm leaks
  – Mimic real execution

```java
public static class MyClass {
    int m;
}
static void foo(MyClass x) {
    ...
}
```
A Running Example

```java
public static class MyClass {
    int m;
}

static void foo(MyClass x) {
    x.m = source();
    MyClass y = new MyClass();
    int u = ~x.m;
    y.m = u;
    if (x != y)
        sink(y.m);
}
```

<table>
<thead>
<tr>
<th>type</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>?</td>
<td>?</td>
</tr>
</tbody>
</table>
public static class MyClass {
    int m;
}
static void foo(MyClass x) {
    x.m = source();
    // getPhoneNumber()
    MyClass y = new MyClass();
    int u = ~x.m;
    y.m = u;
    if (x != y)
        sink(y.m);
}
A Running Example

```java
public static class MyClass {
    int m;
}
static void foo(MyClass x) {
    x.m = source();
    MyClass y = new MyClass();
    int u = ~x.m;
    y.m = u;
    if (x != y)
        sink(y.m);
}
```

**taint:** int

**x:**
- **type:** ?,
- **value:** m
A Running Example

```java
public static class MyClass {
    int m;
}

static void foo(MyClass x) {
    x.m = source();
    MyClass y = new MyClass();
    int u = ~x.m;
    y.m = u;
    if (x != y) {
        sink(y.m);
    }
}
```

**x:**
- **type:** int
- **value:** m

**y:**
- **type:** MyClass
- **value:** m:0

**taint:**
- **type:** int
- **value:** ...

? m
public static class MyClass {
    int m;
}
static void foo(MyClass x) {
    x.m = source();
    MyClass y = new MyClass();
    int u = ~x.m;
    y.m = u;
    if (x != y)
        sink(y.m);
}
A Running Example

```java
public static class MyClass {
    int m;
}
static void foo(MyClass x) {
    x.m = source();
    MyClass y = new MyClass();
    int u = ~x.m;
    y.m = u;
    if (x != y)
        sink(y.m);
}
```

Type and Value:
- **x:** `?` - `m`
- **taint:** `int` - `...`
- **u:** `int` - `...`
- **y:** `MyClass` - `m`
A Running Example

public static class MyClass {
    int m;
}

static void foo(MyClass x) {
    x.m = source();
    MyClass y = new MyClass();
    int u = ~x.m;
    y.m = u;
    if (x != y)
        sink(y.m);
}
A Running Example

```java
public static class MyClass {
    int m;
}

static void foo(MyClass x) {
    x.m = source();
    MyClass y = new MyClass();
    int u = ~x.m;
    y.m = u;
    if (x != y) {
        sink(y.m);
    }
}
```

![Diagram](image-url)
A Running Example

```java
public static class MyClass {
    int m;
}
static void foo(MyClass x) {
    x.m = source();
    MyClass y = new MyClass();
    int u = ~x.m;
    y.m = u;
    if (x != y)
        sink(y.m);
}
```

Sink() meets a taint, leak!
Approximated Execution

• Executing bytecode
  – 15 rules for manipulating unknown objects
Approximated Execution

• Executing bytecode
  – 15 rules for manipulating unknown objects
  – Keep executing until
    • finding a leak
    • function terminated
    • encountering an unknown control flow -> in the paper
Approximated Execution

• Executing bytecode
  – 15 rules for manipulating unknown objects
  – Keep executing until
    • finding a leak
    • function terminated
    • encountering an unknown control flow -> in the paper

✓ No false positives
✓ Deterministic leaks
?
External input dependent leaks
Evaluation

• Correctness & Accuracy
  – Malware genome datasets [S&P’12]
  – Droidbench: well-crafted cases against analysis

• Analysis time & Memory cost
  – Malware and real apps

• Some real leaks found in real apps
Accuracy

- Malware dataset: 1,004 positives + 1 negative
Accuracy

- Malware dataset: 1,004 positives + 1 negative

AppAudit
FlowDroid
AppIntent

FN=0.7%

External input dependent leak
Accuracy

- Malware dataset: 1,004 positives + 1 negative

- FP AppIntent
- FP FlowDroid
- no FP AppAudit

99.3%
The True Negative

Hardcoded Private Key

Custom DES

String

Parse

URL[]
The True Negative

Malformed configuration file disables the malware

Hardcoded Private Key

Custom DES

Parse

Null
Analysis Time

- **ApplIntent**
  - Real app: N/A
  - Malware: > 1hr

- **FlowDroid**
  - Real app: 11.44
  - Malware: 40.54

- **AppAudit**
  - Real app: 0.61
  - Malware: 4.87

**Average analysis time (seconds)**
Memory Cost

<table>
<thead>
<tr>
<th>Tool</th>
<th>Real app (seconds)</th>
<th>Malware (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ApIntent</td>
<td>N/A</td>
<td>&gt; 1hr</td>
</tr>
<tr>
<td>32GB</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FlowDroid</td>
<td>11.44</td>
<td>40.54</td>
</tr>
<tr>
<td>&gt;4GB</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AppAudit</td>
<td>0.61</td>
<td>4.87</td>
</tr>
<tr>
<td>0.2GB</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Not a whole-program analysis
Memory Cost

- **Appltent**: Real app: N/A
- **32GB**: >1hr
- **FlowDroid**: 11.44 seconds
- **>4GB**: 40.54 seconds
- **AppAudit**: 4.87 seconds

*Not a whole-program analysis*

*Only AppAudit is practical for running on a mobile device!*

---

**Average analysis time (seconds)**
400 real apps, 30 data leaks

No false positives
Manual confirmed
# Real Leaks

<table>
<thead>
<tr>
<th>Name</th>
<th>Component</th>
<th>Source</th>
<th>Venue</th>
<th>Privacy Policy</th>
<th>Installs (M for millions)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Texas Poker v4.0.1</td>
<td>App</td>
<td>Location</td>
<td>HTTP GET</td>
<td>×</td>
<td>10M-50M</td>
</tr>
<tr>
<td>Word Search v1.14</td>
<td>Mobfox</td>
<td>Location, IMEI</td>
<td>HTTP GET</td>
<td>app,lib</td>
<td>0.5M-1M</td>
</tr>
<tr>
<td>Speedtest v2.09</td>
<td>Mobfox</td>
<td>Location, IMEI</td>
<td>HTTP GET</td>
<td>app,lib</td>
<td>10M-50M</td>
</tr>
<tr>
<td>Brightest Flashlight v2.3.3</td>
<td>MDotm, Mobclix</td>
<td>IMEI, IMSI</td>
<td>HTTP GET</td>
<td>app,lib</td>
<td>50M-100M</td>
</tr>
</tbody>
</table>
# Real Leaks

<table>
<thead>
<tr>
<th>Name</th>
<th>Component</th>
<th>Source</th>
<th>Venue</th>
<th>Privacy Policy</th>
<th>Installs (M for millions)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Texas Poker v4.0.1</td>
<td>App</td>
<td>Location</td>
<td>HTTP GET</td>
<td>×</td>
<td>10M-50M</td>
</tr>
<tr>
<td>Word Search v1.14</td>
<td>Mobfox</td>
<td>Location, IMEI</td>
<td>HTTP GET</td>
<td>app,lib</td>
<td>0.5M-1M</td>
</tr>
<tr>
<td>Speedtest v2.09</td>
<td>Mobfox</td>
<td>Location, IMEI</td>
<td>HTTP GET</td>
<td>app,lib</td>
<td>10M-50M</td>
</tr>
<tr>
<td>Brightest Flashlight v2.3.3</td>
<td>MDotm, Mobcli</td>
<td>IMEI, IMSI</td>
<td>HTTP GET</td>
<td>app,lib</td>
<td>50M-100M</td>
</tr>
</tbody>
</table>

Total: half billion installs!
# Real Leaks

What component leaks most? Ad libraries

<table>
<thead>
<tr>
<th>Name</th>
<th>Component</th>
<th>Source</th>
<th>Venue</th>
<th>Privacy Policy</th>
<th>Installs (M for millions)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Texas Poker v4.0.1</td>
<td>App</td>
<td>Location</td>
<td>HTTP GET</td>
<td>×</td>
<td>10M-50M</td>
</tr>
<tr>
<td>Word Search v1.14</td>
<td>Mobfox</td>
<td>Location, IMEI</td>
<td>HTTP GET</td>
<td>app, lib</td>
<td>0.5M-1M</td>
</tr>
<tr>
<td>Speedtest v2.09</td>
<td>Mobfox</td>
<td>Location, IMEI</td>
<td>HTTP GET</td>
<td>app, lib</td>
<td>10M-50M</td>
</tr>
<tr>
<td>Brightest Flashlight v2.3.3</td>
<td>MDotm, Mobclix</td>
<td>IMEI, IMSI</td>
<td>HTTP GET</td>
<td>app, lib</td>
<td>50M-100M</td>
</tr>
</tbody>
</table>
# Real Leaks

<table>
<thead>
<tr>
<th>Name</th>
<th>Component</th>
<th>Source</th>
<th>Venue</th>
<th>Privacy Policy</th>
<th>Installs (M for millions)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Texas Poker v4.0.1</td>
<td>App</td>
<td>Location</td>
<td>HTTP GET</td>
<td>×</td>
<td>10M-50M</td>
</tr>
<tr>
<td>Word Search v1.14</td>
<td>Mobfox</td>
<td>Location, IMEI</td>
<td>HTTP GET</td>
<td>app,lib</td>
<td>0.5M-1M</td>
</tr>
<tr>
<td>Speedtest v2.09</td>
<td>Mobfox</td>
<td>Location, IMEI</td>
<td>HTTP GET</td>
<td>app,lib</td>
<td>10M-50M</td>
</tr>
<tr>
<td>Brightest Flashlight v2.3.3</td>
<td>MDotm, Mobclix</td>
<td>IMEI, IMSI</td>
<td>HTTP GET</td>
<td>app,lib</td>
<td>50M-100M</td>
</tr>
</tbody>
</table>

Privacy Policy? Only a few mention 3\textsuperscript{rd} party libs
# Real Leaks

<table>
<thead>
<tr>
<th>Name</th>
<th>Component</th>
<th>Source</th>
<th>Venue</th>
<th>Privacy Policy</th>
<th>Installs (M for millions)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Texas Poker v4.0.1</td>
<td>App</td>
<td>Location</td>
<td>HTTP GET</td>
<td>×</td>
<td>10M-50M</td>
</tr>
<tr>
<td>Word Search v1.14</td>
<td>Mobfox</td>
<td>Location, IMEI</td>
<td>HTTP GET</td>
<td>app,lib</td>
<td>0.5M-1M</td>
</tr>
<tr>
<td>Speedtest v2.09</td>
<td>Mobfox</td>
<td>Location, IMEI</td>
<td>HTTP GET</td>
<td>app,lib</td>
<td>10M-50M</td>
</tr>
<tr>
<td>Brightest Flashlight v2.3.3</td>
<td>MDotm, Mobclix</td>
<td>IMEI, IMSI</td>
<td>HTTP GET</td>
<td>app,lib</td>
<td>50M-100M</td>
</tr>
</tbody>
</table>

What data are leaked most? IMEI, IMSI, Location

Among ~15 taints we tracked
## Real Leaks

<table>
<thead>
<tr>
<th>Name</th>
<th>Component</th>
<th>Source</th>
<th>Venue</th>
<th>Privacy Policy</th>
<th>Installs (M for millions)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Texas Poker v4.0.1</td>
<td>App</td>
<td>Location</td>
<td>HTTP GET</td>
<td>×</td>
<td>10M-50M</td>
</tr>
<tr>
<td>Word Search v1.14</td>
<td>Mobfox</td>
<td>Location, IMEI</td>
<td>HTTP GET</td>
<td>app,lib</td>
<td>0.5M-1M</td>
</tr>
<tr>
<td>Speedtest v2.09</td>
<td>Mobfox</td>
<td>Location, IMEI</td>
<td>HTTP GET</td>
<td>app,lib</td>
<td>10M-50M</td>
</tr>
<tr>
<td>Brightest Flashlight v2.3.3</td>
<td>MDotm, Mobclix</td>
<td>IMEI, IMSI</td>
<td>HTTP GET</td>
<td>app,lib</td>
<td>50M-100M</td>
</tr>
</tbody>
</table>

**How are data leaked?**

HTTP (not encrypted!) and HTTPs
400 real apps, 30 data leaks
http://appaudit.io

400 real apps, 30 data leaks

~140,000 8,537
http://appaudit.io

400 real apps, 30 data leaks

~140,000 8,537

2 days
Work-in-progress

• More auditing details
  – E.g. when the leak would happen
• Mobile version
• Developer tool
  – Android Studio
  – Embed for Github repo
Thank you

http://appaudit.io

<table>
<thead>
<tr>
<th>AppAudit</th>
<th>Developers</th>
<th>Research</th>
<th>Contact</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>App</th>
<th>GooglePlay Rating</th>
<th>Leak phone identity</th>
<th>Read SMS messages</th>
<th>Automatically start</th>
<th>Leak phone identity</th>
<th>Detect virus content</th>
</tr>
</thead>
<tbody>
<tr>
<td>App1</td>
<td>3.6</td>
<td>via a public file on the SDCARD</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>App2</td>
<td>4.5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>App3</td>
<td>3.7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Common Q/A

1. You have FNs. So it is not complete?
2. The definition of leaks?
3. iOS?
4. Native code?
5. What about paid apps?
6. Can AppAudit actually run on a phone?
7. What can I do if I know an app is leaking something?
8. Reflection, control-flow taints, taint sanitization, infinity, multi-threading, compiler sensitivity, file taints, how it actually works?

In the paper
Completeness

- no false positive
  - Generative model
  - AppAudit execution path
  - -> User inputs
  - -> Real execution path
  - that leaks (TaintDroid?)

- no false negative
  - Largely depend on the completeness model itself
Completeness (cont.)

App 1

Id=....

OCR the text

Take screen snapshot

App 2

Inter-process, computer vision based leak
Completeness (cont.)

App 1

Id=....

App 3

Front camera: take picture

Human-phone, computer vision based leak