Collaborative Verification of Information Flow for a High-Assurance App Store

Michael D. Ernst, René Just, Suzanne Millstein, Werner Dietl*, Stuart Pernsteiner, Franziska Roesner, Karl Koscher, Paulo Barros, Ravi Bhoraskar, Seungyeop Han, Paul Vines, and Edward X. Wu

University of Washington

*University of Waterloo

November 6, 2014
Current commercial app stores

Several hundred new apps per day

Approval process
Current commercial app stores

Several hundred new apps per day

Approval process

Problem: Every major app store has approved malware!
Current commercial app stores

Several hundred new apps per day

Approval process

Problem: Every major app store has approved malware!

Best-effort solution: Malware removed when encountered
High-assurance app stores

Needed in multiple domains
- Government app stores (e.g., DoD)
- Corporate app stores (e.g., financial sector)
- App stores for medical apps

Require stronger guarantees
- Verified absence of (certain types of) malware

Verification is costly
- Effort is solely on app store side
- Analyst needs to understand/reverse-engineer the app
High-assurance app stores

Needed in multiple domains
- Government app stores (e.g., DoD)
- Corporate app stores (e.g., financial sector)
- App stores for medical apps

Require stronger guarantees
- Verified absence of (certain types of) malware

Verification is costly
- Effort is solely on app store side
- Analyst needs to understand/reverse-engineer the app

Our solution: Collaboratively verify absence of malware
Our focus: Information-flow malware
Example: Information-flow malware

<table>
<thead>
<tr>
<th>App</th>
<th>Permissions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sudoku</td>
<td>Read location, Internet</td>
</tr>
</tbody>
</table>
Example: Information-flow malware

App
Sudoku

Permissions
Read location
Internet
Example: Information-flow malware

App
Sudoku

Permissions
Read location
Internet

App
Camera

Permissions
Read location
Internet
Example: Information-flow malware

<table>
<thead>
<tr>
<th>App</th>
<th>Permissions</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Sudoku</td>
<td>Read location</td>
<td></td>
</tr>
<tr>
<td>Camera</td>
<td>Read location</td>
<td></td>
</tr>
</tbody>
</table>

App: Sudoku and Camera
Permissions: Read location, Internet
Example: Information-flow malware

<table>
<thead>
<tr>
<th>App</th>
<th>Permissions</th>
<th>Information flow</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sudoku</td>
<td>Read location</td>
<td>Location → Internet</td>
</tr>
<tr>
<td>Camera</td>
<td>Read location</td>
<td>Internet</td>
</tr>
</tbody>
</table>

App Permissions Information flow

- Sudoku
  - Permissions: Read location
  - Information flow: Location → Internet
- Camera
  - Permissions: Read location
  - Information flow: Internet
Example: Information-flow malware

<table>
<thead>
<tr>
<th>App</th>
<th>Permissions</th>
<th>Information flow</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sudoku</td>
<td>Read location, Internet</td>
<td>Internet → Location</td>
</tr>
<tr>
<td>Camera</td>
<td>Read location, Internet</td>
<td>Internet → Internet</td>
</tr>
</tbody>
</table>
## Example: Information-flow malware

<table>
<thead>
<tr>
<th>App</th>
<th>Permissions</th>
<th>Information flow</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sudoku</td>
<td>Read location Internet</td>
<td></td>
</tr>
<tr>
<td>Camera</td>
<td>Read location Internet</td>
<td>Location → Internet</td>
</tr>
<tr>
<td>Map</td>
<td>Read location Internet</td>
<td></td>
</tr>
</tbody>
</table>
Example: Information-flow malware

<table>
<thead>
<tr>
<th>App</th>
<th>Permissions</th>
<th>Information flow</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sudoku</td>
<td>Read location</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Internet</td>
<td></td>
</tr>
<tr>
<td>Camera</td>
<td>Read location</td>
<td>Location →</td>
</tr>
<tr>
<td></td>
<td>Internet</td>
<td>Internet</td>
</tr>
<tr>
<td>Map</td>
<td>Read location</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Internet</td>
<td></td>
</tr>
</tbody>
</table>
Example: Information-flow malware

- **App**
  - Sudoku
  - Camera
  - Map

- **Permissions**
  - Read location
  - Internet

- **Information flow**
  - Location →
  - Internet

René Just, UW CSE
Collaborative Verification of Information Flow for a High-Assurance App Store
### Example: Information-flow malware

<table>
<thead>
<tr>
<th>App</th>
<th>Permissions</th>
<th>Information flow</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sudoku</td>
<td>Read location → Internet</td>
<td>![Malware Icon]</td>
</tr>
<tr>
<td>Camera</td>
<td>Read location → Internet</td>
<td>![Malware Icon]</td>
</tr>
<tr>
<td>Map</td>
<td>Read location → Internet</td>
<td>![Malware Icon]</td>
</tr>
</tbody>
</table>
### Example: Information-flow malware

<table>
<thead>
<tr>
<th>App</th>
<th>Permissions</th>
<th>Information flow</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sudoku</td>
<td>Read location Internet</td>
<td></td>
</tr>
<tr>
<td>Camera</td>
<td>Read location Internet</td>
<td>Location → Internet</td>
</tr>
<tr>
<td>Map</td>
<td>Read location Internet</td>
<td>Location → BadGuy.com</td>
</tr>
</tbody>
</table>
Example: Information-flow malware

<table>
<thead>
<tr>
<th>App</th>
<th>Permissions</th>
<th>Information Flow</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sudoku</td>
<td>Read location</td>
<td>Internet</td>
</tr>
<tr>
<td>Camera</td>
<td>Read location</td>
<td>Internet</td>
</tr>
<tr>
<td>Map</td>
<td>Read location</td>
<td>Location → Internet</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Location → BadGuy.com</td>
</tr>
</tbody>
</table>
Example: Information-flow malware

Prevent malware using an information flow type-system

Sudoku

App Permissions Information flow

Read location Internet

Camera

Read location Internet

Map

Read location Internet

Location → Internet

Location → BadGuy.com
Approach: Overview

Collaborative verification model
  ▶ Leverage but don’t trust the developer

Information Flow Type-checker (IFT)
  ▶ Finer-grained permission model for Android
  ▶ False positives and declassifications
  ▶ Implicit information flow

Evaluation
  ▶ Effectiveness: Effective for real malware in real apps
  ▶ Usability: Low annotation and auditing burden
Collaborative verification model

Developer provides

- App description
- Annotated source code
Collaborative verification model

Developer provides

- App description
- Information flow policy
- Annotated source code
- Declassification justifications

High-level description of information flows in app
(LOCATION -> INTERNET)
Collaborative verification model

Developer provides

- App description
- Information flow policy
- Annotated source code
- Declassification justifications

App store verifies
Collaborative verification model

Developer provides

- App description
- Information flow policy
- Annotated source code
- Declassification justifications

1. Analyst verifies: acceptable behavior

App store verifies
Collaborative verification model

Developer provides

App description

Information flow policy

Annotated source code

Declassification justifications

1 Analyst verifies: acceptable behavior

2 Type checker verifies: annotations consistent

App store verifies
Collaborative verification model

Developer provides

- App description
- Information flow policy
- Annotated source code
- Declassification justifications

App store verifies

1. Analyst verifies: acceptable behavior
2. Type checker verifies: annotations consistent
3. Analyst verifies: declassifications

René Just, UW CSE Collaborative Verification of Information Flow for a High-Assurance App Store
Collaborative verification model

Developer provides

App description  →  Information flow policy  ↔  Annotated source code  →  Declassification justifications

Analyst verifies: acceptable behavior

Type checker verifies: annotations consistent

Analyst verifies: declassifications

App store verifies

Developer and analyst do tasks that are easy for them
Verification of information flow

Information flow policy

Annotated source code

Type checker verifies: annotations consistent
Verification of information flow

Type checker verifies:
annotations consistent

Information flow policy

Annotated source code
Information flow policy

High-level description of permitted information flows

- READ_SMS -> INTERNET
- READ_CLIPBOARD -> DISPLAY
- USER_INPUT -> CALL_PHONE
- ACCESS_FINE_LOCATION -> INTERNET(maps.google.com)
# Information flow policy

## High-level description of permitted information flows

<table>
<thead>
<tr>
<th>Source</th>
<th>flows to</th>
<th>Sink</th>
</tr>
</thead>
<tbody>
<tr>
<td>READ_SMS</td>
<td>-&gt;</td>
<td>INTERNET</td>
</tr>
<tr>
<td>READ_CLIPBOARD</td>
<td>-&gt;</td>
<td>DISPLAY</td>
</tr>
<tr>
<td>USER_INPUT</td>
<td>-&gt;</td>
<td>CALL_PHONE</td>
</tr>
<tr>
<td>ACCESS_FINE_LOCATION</td>
<td>-&gt;</td>
<td>INTERNET(maps.google.com)</td>
</tr>
</tbody>
</table>
Information flow policy

High-level description of permitted information flows

<table>
<thead>
<tr>
<th>Source</th>
<th>flows to</th>
<th>Sink</th>
</tr>
</thead>
<tbody>
<tr>
<td>READ_SMS</td>
<td>-&gt;</td>
<td>INTERNET</td>
</tr>
<tr>
<td>READ_CLIPBOARD</td>
<td>-&gt;</td>
<td>DISPLAY</td>
</tr>
<tr>
<td>USER_INPUT</td>
<td>-&gt;</td>
<td>CALL_PHONE</td>
</tr>
<tr>
<td>ACCESS_FINE_LOCATION</td>
<td>-&gt;</td>
<td>INTERNET(maps.google.com)</td>
</tr>
</tbody>
</table>

Sources and Sinks

- Default Android permissions (145)

Not sufficient to model information flow!
### Information flow policy

#### High-level description of permitted information flows

<table>
<thead>
<tr>
<th>Source</th>
<th>flows to</th>
<th>Sink</th>
</tr>
</thead>
<tbody>
<tr>
<td>READ_SMS</td>
<td></td>
<td>INTERNET</td>
</tr>
<tr>
<td>READ_CLIPBOARD</td>
<td></td>
<td>DISPLAY</td>
</tr>
<tr>
<td>USER_INPUT</td>
<td></td>
<td>CALL_PHONE</td>
</tr>
<tr>
<td>ACCESS_FINE_LOCATION</td>
<td></td>
<td>INTERNET(maps.google.com)</td>
</tr>
</tbody>
</table>

#### Sources and Sinks

- Default Android permissions (145)
- **Additional sensitive resources** (28)
Information flow policy

High-level description of permitted information flows

<table>
<thead>
<tr>
<th>Source</th>
<th>flows to</th>
<th>Sink</th>
</tr>
</thead>
<tbody>
<tr>
<td>READ_SMS</td>
<td>-&gt;</td>
<td>INTERNET</td>
</tr>
<tr>
<td>READ_CLIPBOARD</td>
<td>-&gt;</td>
<td>DISPLAY</td>
</tr>
<tr>
<td>USER_INPUT</td>
<td>-&gt;</td>
<td>CALL_PHONE</td>
</tr>
<tr>
<td>ACCESS_FINE_LOCATION</td>
<td>-&gt;</td>
<td>INTERNET(maps.google.com)</td>
</tr>
</tbody>
</table>

Sources and Sinks

- Default Android permissions (145)
- Additional sensitive resources (28)
- **Parameterized permissions**
Verification of information flow

Type checker verifies: annotations consistent
Information flow types: sources and sinks

@Source Where might a value come from?
@Sink Where might a value flow to?
Information flow types: sources and sinks

@Source Where might a value come from?
@Sink Where might a value flow to?

Android API

```java
void sendToInternet(String message);
String readGPS();
```
Information flow types: sources and sinks

**@Source** Where might a value come from?
**@Sink** Where might a value flow to?

Android API

```java
void sendToInternet(String message);
String readGPS();
```

To Internet
Information flow types: sources and sinks

@Source Where might a value come from?
@Sink Where might a value flow to?

Android API

```java
void sendToInternet(@Sink(INTERNET)String message);
String readGPS();
```
Information flow types: sources and sinks

**@Source** Where might a value come from?

**@Sink** Where might a value flow to?

```java
void sendToInternet(@Sink(INTERNET) String message);
String readGPS();
```

Android API

From Location
Information flow types: sources and sinks

@Source Where might a value come from?
@Sink Where might a value flow to?

Android API

```java
void sendToInternet(@Sink(INTERNET)String message);
@Source(LOCATION)String readGPS();
```
Information flow types: sources and sinks

@Source Where might a value come from?
@Sink Where might a value flow to?

Android API
void sendToInternet(@Sink(INTERNET) String message);
@Source(LOCATION) String readGPS();

App code
String loc = readGPS();
sendToInternet(loc);
Information flow types: sources and sinks

@Source Where might a value come from?
@Sink Where might a value flow to?

Android API

```java
void sendToInternet(@Sink(INTERNET) String message);
@Source(LOCATION) String readGPS();
```

App code

```java
@Source(LOCATION)@Sink(INTERNET) String loc = readGPS();
sendToInternet(loc);
```
Information flow types: sources and sinks

@Source Where might a value come from?
@Sink Where might a value flow to?

<table>
<thead>
<tr>
<th>Android API</th>
<th>API annotations are pre-verified</th>
</tr>
</thead>
<tbody>
<tr>
<td>void sendToInternet(@Sink(INTERNET)String message);</td>
<td></td>
</tr>
<tr>
<td>@Source(LOCATION)String readGPS();</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>App code</th>
<th>Developer annotations are not trusted</th>
</tr>
</thead>
<tbody>
<tr>
<td>@Source(LOCATION)@Sink(INTERNET)String loc = readGPS();</td>
<td></td>
</tr>
<tr>
<td>sendToInternet(loc);</td>
<td></td>
</tr>
</tbody>
</table>
Type hierarchy for sources and sinks

@Source(ANY)
@Source({SMS, LOCATION})
@Source(SMS)
@Source(LOCATION)
@Source({})

@Sink({})
@Sink(INTERNET)
@Sink(SMS)
@Sink({INTERNET, SMS})
@Sink(ANY)
Type hierarchy for sources and sinks

@Source(ANY) ≡ @Source({SMS, LOCATION, INTERNET, ...})
Type hierarchy for sources and sinks

```
@Source(SMS) String sms = ...;
@Source({SMS, LOCATION}) String smsLoc = sms;
```
Type hierarchy for sources and sinks

@Source(ANY)
@Source({SMS, LOCATION})
@Source(SMS)
@Source(LOCATION)
@Source({})

@Sink({})
@Sink(INTERNET)
@Sink(SMS)
@Sink({INTERNET, SMS})
@Sink(ANY)

@Source(SMS) String sms = ...;
@Source(LOCATION) String loc = sms;
Type hierarchy for sources and sinks
Type hierarchy for sources and sinks

@Source(ANY)
@Source({SMS, LOCATION})
@Source(SMS)
@Source(LOCATION)
@Source({})

@Sink({})
@Sink(INTERNET)
@Sink(SMS)
@Sink({INTERNET, SMS})
@Sink({INTERNET, SMS})
@Sink(ANY)

@Sink({INTERNET, SMS}) String toInetSms;
@Sink(SMS) String toSms = toInetSms;
Type hierarchy for sources and sinks

@Source(ANY)
@Source({SMS, LOCATION})
@Source(SMS)
@Source(LOCATION)
@Source({})

@Sink({})
@Sink(INTERNET)
@Sink(SMS)
@Sink({INTERNET, SMS})
@Sink(ANY)

@Sink(SMS) String toSms;
@Sink(INTERNET) String toInet = toSms;
Verification of information flow

Information flow policy

Annotated source code

Type checker verifies: annotations consistent
Information Flow Type-checker (IFT): Overview

Guarantees of type-checking

1. Annotations are consistent with code (type correctness)
2. Annotations are consistent with flow policy

Type checker verifies:
annotations consistent
Information Flow Type-checker (IFT): Overview

Guarantees of type-checking

1. Annotations are consistent with code (type correctness)
2. Annotations are consistent with flow policy

Type checker verifies: annotations consistent

No undisclosed information flows in app
Information Flow Type-checker (IFT): Example

App code

```
@Source(LOCATION)@Sink(INTERNET) String loc = readGPS();
sendToInternet(loc);
```

Flow policy

LOCATION -> INTERNET

Type checker verifies: annotations consistent
Information Flow Type-checker (IFT): Example

App code

```java
@Source(LOCATION)@Sink(INTERNET) String loc = readGPS();
sendToInternet(loc);
```

Flow policy

LOCATION -> INTERNET

Type checker verifies:
annotations consistent
Information Flow Type-checker (IFT): Example

App code

```java
@Source(LOCATION)@Sink(INTERNET) String loc = readGPS();
sendSms(loc);
```

Flow policy
LOCATION -> INTERNET

Type checker verifies: annotations consistent
Information Flow Type-checker (IFT): Example

App code

```java
@Source(LOCATION)@Sink(INTERNET)
String loc = readGPS();
sendSms(loc);
```

Flow policy

LOCATION -> INTERNET

Type checker verifies:
annotations consistent

Incompatible sinks:
INTERNET ≠: SMS
Information Flow Type-checker (IFT): Example

App code

```java
@Source(LOCATION) @Sink(SMS)
String loc = readGPS();
sendSms(loc);
```

Flow policy

LOCATION -> INTERNET

Type checker verifies: annotations consistent
Information Flow Type-checker (IFT): Example

App code

```java
@Source(LOCATION)@Sink(SMS) String loc = readGPS();
sendSms(loc);
```

Flow policy

LOCATION → INTERNET

Type checker verifies: annotations consistent

Forbidden flow:

LOCATION → SMS
False positives and declassifications

App code

```java
@Source({LOCATION, SMS}) String[] array;
array[0] = readGPS();
array[1] = readSMS();

@Source(LOCATION) String loc = array[0];
```
False positives and declassifications

<table>
<thead>
<tr>
<th>App code</th>
</tr>
</thead>
</table>
| ```
@Source({LOCATION, SMS}) String [] array;
array[0] = readGPS();
array[1] = readSMS();
@Source(LOCATION) String loc = array[0];
``` |

Developer can suppress false-positive warnings.
App store employee verifies each declassification.
False positives and declassifications

App code

```java
@Source({LOCATION, SMS}) String [] array;
array[0] = readGPS();
array[1] = readSMS();

@Source(LOCATION) String loc = array[0];
```

- Developer can suppress false-positive warnings
- App store employee verifies each declassification
False positives and declassifications

App code

```
@Source({LOCATION, SMS}) String[] array;
array[0] = readGPS();
array[1] = readSMS();
@SuppressWarnings("flow") // Safe: returns location data
@Source(LOCATION) String loc = array[0];
```

Declasifications

- Developer can suppress false-positive warnings
- App store employee verifies each declassification
Reducing false positives

Flow sensitivity

- Type refinement with intra-procedural data flow analysis

App code

```java
@Source({LOCATION, SMS}) String value;
if (...) {
    value = readSMS();
    ... value: @Source(SMS)
}
... value: @Source({LOCATION, SMS})
```
Reducing false positives

Flow sensitivity

- Type refinement with intra-procedural data flow analysis

Context sensitivity

- Polymorphism (e.g., String operations, I/O streams, etc.)

App code

```java
@Source({LOCATION, SMS}) String value = ...;
String substring = value.substring(0,8);
```

Returns `@Source({LOCATION, SMS})`
Reducing false positives

Flow sensitivity
  ▶ Type refinement with intra-procedural data flow analysis

Context sensitivity
  ▶ Polymorphism (e.g., String operations, I/O streams, etc.)

Indirect control flow
  ▶ Constant value propagation
  ▶ Reflection analysis
  ▶ Intent analysis
Implicit information flow

```java
@Source(USER_INPUT) long creditCard = getCard();
long i=0;
while (true) {
    if (++i == creditCard) {
        sendToInternet(i);
    }
}
```

App code
Implicit information flow

App code

```java
@Source(USER_INPUT)long creditCard = getCard();
long i=0;
while (true) {
    if (++i == creditCard) {
        sendToInternet(i);
    }
}
```

Card number implicitly leaked

Classic approach (Denning and Denning, CACM’77)

- Taint all computations in dynamic scope
- Over-tainting may lead to taint explosion
Implicit information flow

App code

```java
@Source(USER_INPUT) long creditCard = getCard();
long i=0;
while (true) {
    if (++i == creditCard) {
        sendToInternet(i);
    }
}
```

Our approach: Prune irrelevant conditions

- Add additional sink **CONDITIONAL**
- Type-checker warning for conditions with sensitive source
Implicit information flow

App code

```java
@Source(USER_INPUT)long creditCard = getCard();
long i=0;
while (true) {
    if (++i == creditCard) {
        sendToInternet(i);
    }
}
```

Our approach: Prune irrelevant conditions

- Add additional sink **CONDITIONAL**
- Type-checker warning for conditions with sensitive source

Analyst must manually verify

- Analyst is aware of context
- No need to analyze dynamic scope for irrelevant conditions (e.g., null checks, malicious conditions, or trigger)
Evaluation: Overview

Are our permission model and type system effective?
- Adversarial Red Team challenge
- Evaluation of effectiveness for real malware

Is our approach effective and efficient in a time-constrained set up?
- Control team study
- Comparison of effectiveness and efficiency to control team

Is our verification model applicable for real-world apps?
- Usability study with annotators and auditors
- Evaluation of annotation and auditing burden
Evaluation: Overview

Are our permission model and type system effective?
- Adversarial Red Team challenge
- Evaluation of effectiveness for real malware

Is our approach effective and efficient in a time-constrained set up?
- Control team study
- Comparison of effectiveness and efficiency to control team

Is our verification model applicable for real-world apps?
- Usability study with annotators and auditors
- Evaluation of annotation and auditing burden

Apps are not pre-annotated
Adversarial Red Team challenge

Setup

- 5 independent Red Teams
- 72 Android apps (47 malicious with information-flow malware)
- 8,000 LOC and 12 permissions per app on average
Adversarial Red Team challenge

Setup

- 5 independent Red Teams
- 72 Android apps (47 malicious with information-flow malware)
- 8,000 LOC and 12 permissions per app on average

Results for 47 malicious apps

- 96% overall detection rate — 4% require modeling of information flow paths (LOCATION -> ENCRYPT -> INTERNET)
- 60% of apps require our finer-grained sources and sinks
Control team study

Setup

- Control team using dynamic and static analysis tools
- 18 Android apps (13 malicious)
- 7,000 LOC and 16 permissions per app on average
Control team study

Setup

- Control team using dynamic and static analysis tools
- 18 Android apps (13 malicious)
- 7,000 LOC and 16 permissions per app on average

Results

![Graph showing detection rate and analysis time]

Detection rate

Analysis time

Ratio in %
Usability study

Setup

- 2 groups acting as annotators and auditors
- 11 Android apps (1 malicious)
- 900 LOC and 12 permissions per app on average
Usability study

Setup

- 2 groups acting as annotators and auditors
- 11 Android apps (1 malicious)
- 900 LOC and 12 permissions per app on average

Annotation burden

- 96% of type annotations are inferred
- Annotations required: 6 per 100 lines of code
- Annotation time: 16 minutes per 100 lines of code

Most time spent on reverse engineering
Usability study

Declassifications

- 50% of apps had no declassifications
- On average 3 declassification per 1,000 lines of code

IFT’s features effectively reduce false positives
Usability study

Declassifications

- 50% of apps had no declassifications
- On average 3 declassification per 1,000 lines of code

IFT’s features effectively reduce false positives

Auditing burden

- Overall review time: 3 minutes per 100 lines of code
- 35% of time: review the flow policy
- 65% of time: review declassifications & conditionals

Only 23% of conditionals needed to be reviewed
Related work: Information flow

**Jif** *(Myers, POPL’99)*
- A security-typed language (incompatible Java extension)
- Supports dynamic checks and focuses on expressiveness

**FlowDroid** *(Arzt et al., PLDI’14), SuSi** *(Rasthofer et al., NDSS’14)*
- FlowDroid propagates sources and sinks found by SuSi
- SuSi classifies Android API methods using machine learning

**IFT makes static verification of Android apps practical**
- Finer-grained sources and sinks at type level
- Compiler plug-in using standard Java type annotations
Related work: Collaborative verification model

Verifying browser extensions

- IBEX *(Guha et al., S&P’11)*
  - Verification of Fine (ML dialect) against complex policies
- Lerner *et al.*, ESORICS’13
  - Verification of private browsing using annotated JavaScript

IFT verifies information flow in Android apps using a high-level flow policy

Automated policy verification

- Crowd-sourcing *(Agarwal & Hall, MobiSys’13)*
- Natural language processing *(Pandita et al., USENIX’13)*
- Clustering *(Gorla et al., ICSE’14)*

Could aid manual verification of flow policies
Conclusions

Collaborative verification model
- Low overall verification effort for developer and app store analyst
- IFT combined with other analyses

Information Flow Type-checker (IFT)
- Context and flow-sensitive type system
- Fine-grained model for sources and sinks
- High-level information flow policy

Evaluation
- Detected 96% information-flow malware
- Low annotation and auditing burden
- Low false-positive rate

https://www.cs.washington.edu/sparta