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

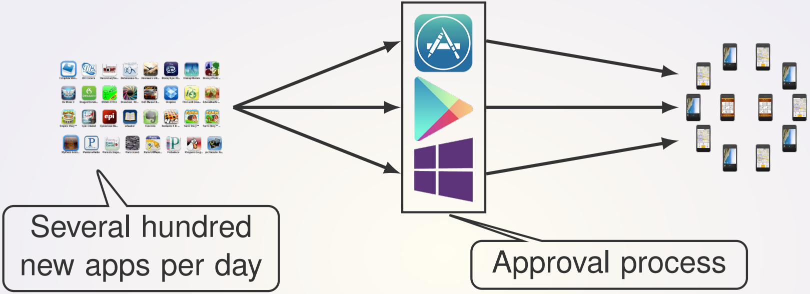
Michael D. Ernst, **René Just**, Suzanne Millstein, Werner Dietl*, Stuart Pernsteiner, Franziska Roesner, Karl Koscher, Paulo Barros, Ravi Bhoraskar, Seungyeop Han, Paul Vines, and Edward X. Wu

University of Washington

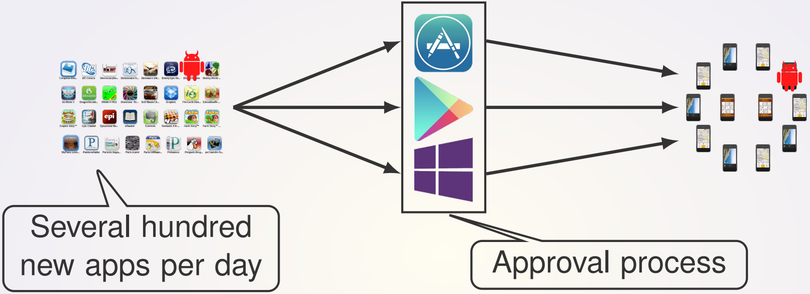
*University of Waterloo

November 6, 2014

Current commercial app stores

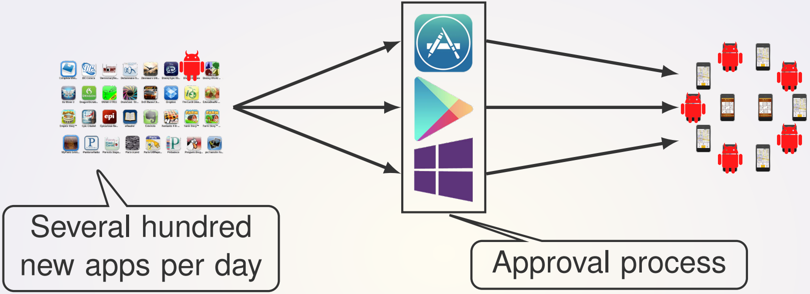


Current commercial app stores



Problem: Every major app store has approved malware!

Current commercial app stores



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

App

Sudoku

Permissions

Read location
Internet

Example: Information-flow malware

App

Sudoku

Permissions

Read location

Internet

Example: Information-flow malware

App	Permissions
 Sudoku	Read location Internet
 Camera	Read location Internet

Example: Information-flow malware

App	Permissions
 Sudoku	Read location Internet
 Camera	Read location Internet

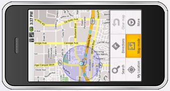
Example: Information-flow malware

App	Permissions	Information flow
 Sudoku	Read location Internet	
 Camera	Read location Internet	Location → Internet

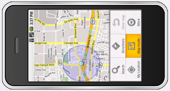
Example: Information-flow malware

App	Permissions	Information flow
 Sudoku	Read location Internet	
 Camera	Read location Internet	Location → Internet

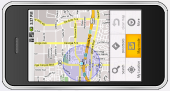
Example: Information-flow malware

App	Permissions	Information flow
 Sudoku	Read location Internet	
 Camera	Read location Internet	Location → Internet
 Map	Read location Internet	

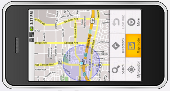
Example: Information-flow malware

App	Permissions	Information flow
 Sudoku	Read location Internet	
 Camera	Read location Internet	Location → Internet
 Map	Read location Internet	

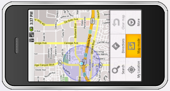
Example: Information-flow malware

App	Permissions	Information flow
 Sudoku	Read location Internet	
 Camera	Read location Internet	Location → Internet
 Map	Read location Internet	Location → Internet

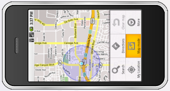
Example: Information-flow malware

App	Permissions	Information flow
 Sudoku	Read location Internet	
 Camera	Read location Internet	Location → Internet
 Map	Read location Internet	Location → Internet

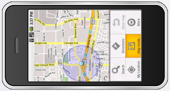
Example: Information-flow malware

App	Permissions	Information flow	
 Sudoku	Read location Internet		
 Camera	Read location Internet	Location → Internet	
 Map	Read location Internet	Location → Internet	Location → BadGuy.com

Example: Information-flow malware

App	Permissions	Information flow	
 Sudoku	**Read location** **Internet**		
 Camera	**Read location** **Internet**	**Location → Internet**	
 Map	**Read location** **Internet**	**Location → Internet**	**Location → BadGuy.com**

Example: Information-flow malware

App	Permissions	Information flow	
 Sudoku	Read location Internet		
 Camera	Read location Internet	Location → Internet	
 Map	Read location Internet	Location → Internet	Location → BadGuy.com

Prevent malware using an information flow type-system

Approach: Overview

Collaborative verification model

- Leverage but don't trust the developer

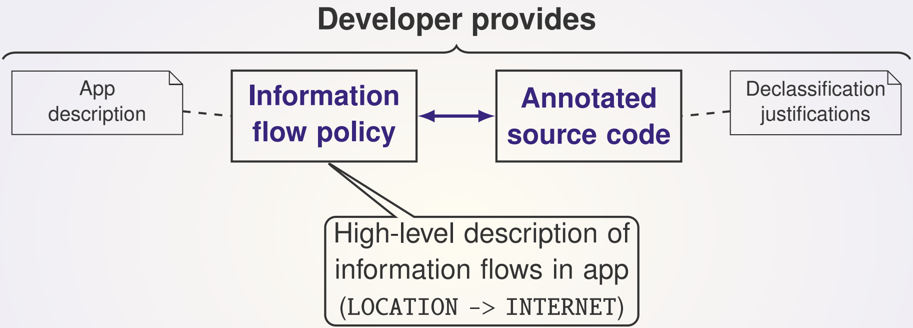
Information Flow Type-checker (IFT)

- Finer-grained permission model for Android
- False positives and declassifications
- Implicit information flow

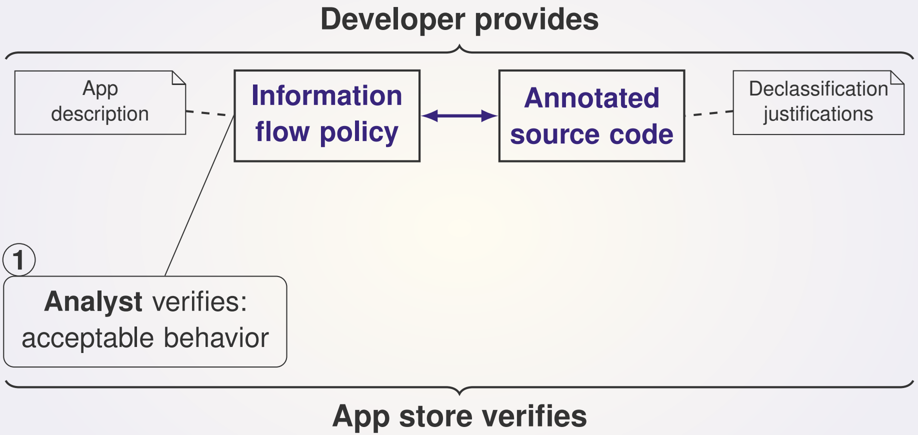
Evaluation

- Effectiveness: Effective for real malware in real apps
- Usability: Low annotation and auditing burden

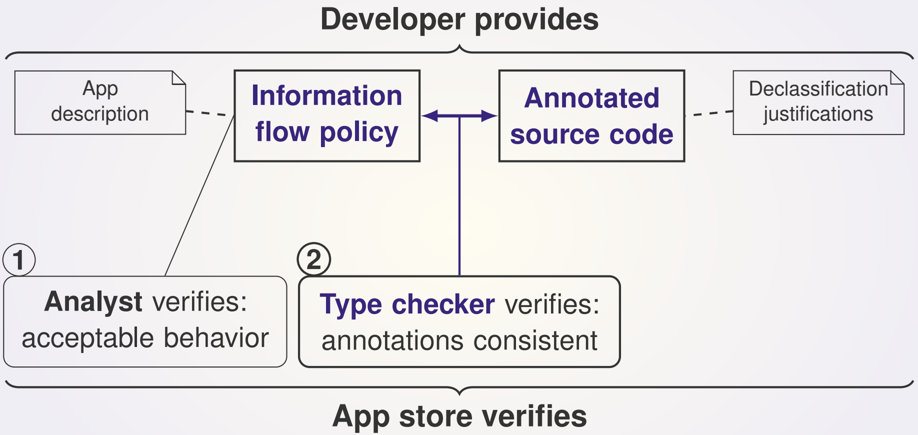
Collaborative verification model



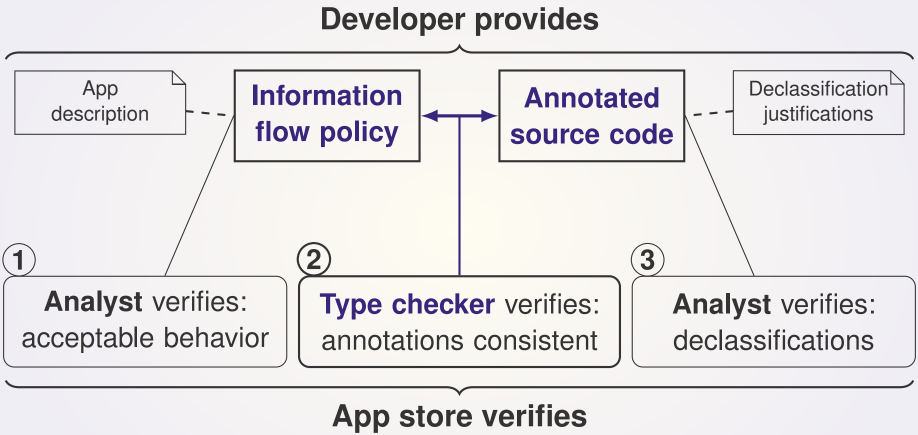
Collaborative verification model



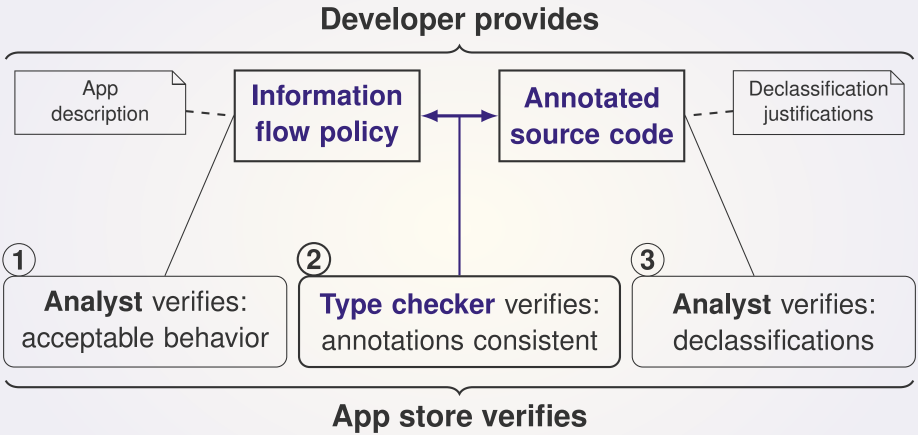
Collaborative verification model



Collaborative verification model

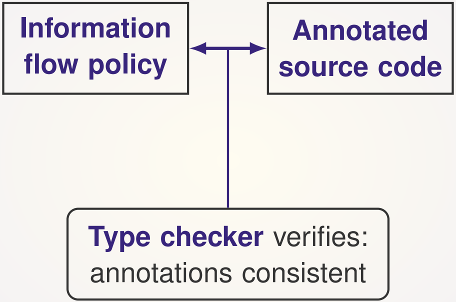


Collaborative verification model

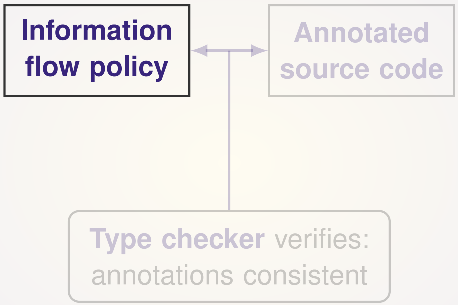


Developer and analyst do tasks that are easy for them

Verification of information flow



Verification of information flow



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

Source	flows to	Sink
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

Source	flows to	Sink
READ_SMS	->	**INTERNET**
READ_CLIPBOARD	->	DISPLAY
USER_INPUT	->	**CALL_PHONE**
ACCESS_FINE_LOCATION	->	INTERNET(maps.google.com)

Sources and Sinks

- **Default Android permissions (145)**

Not sufficient to model information flow!

Information flow policy

High-level description of permitted information flows

Source	flows to	Sink
READ_SMS	->	INTERNET
READ_CLIPBOARD	->	**DISPLAY**
USER_INPUT	->	CALL_PHONE
ACCESS_FINE_LOCATION	->	INTERNET(maps.google.com)

Sources and Sinks

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

Information flow policy

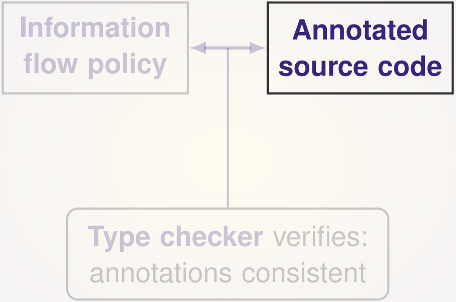
High-level description of permitted information flows

Source	flows to	Sink
`READ_SMS`	`->`	`INTERNET`
`READ_CLIPBOARD`	`->`	`DISPLAY`
`USER_INPUT`	`->`	`CALL_PHONE`
`ACCESS_FINE_LOCATION`	`->`	**`INTERNET(maps.google.com)`**

Sources and Sinks

- Default Android permissions (145)
- Additional sensitive resources (28)
- **Parameterized permissions**

Verification of information flow



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

```
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?

To Internet

Android API

```
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

```
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?

Android API

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

From Location

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();
```

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

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

App code

```
@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?

Android API

API annotations are pre-verified

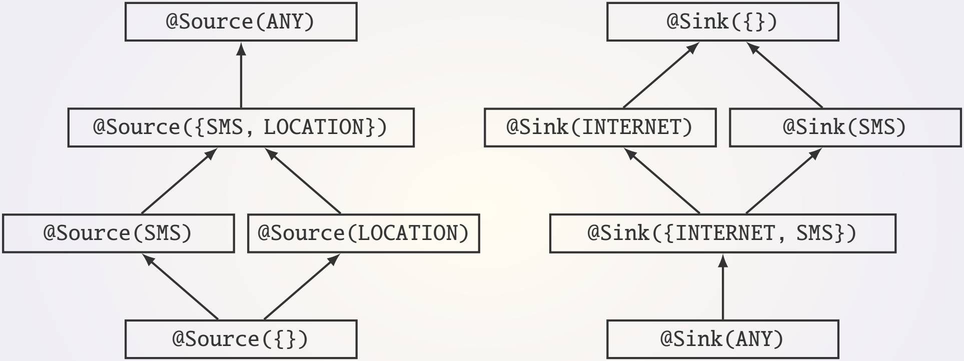
```
void sendToInternet(@Sink(INTERNET)String message);
@Source(LOCATION)String readGPS();
```

App code

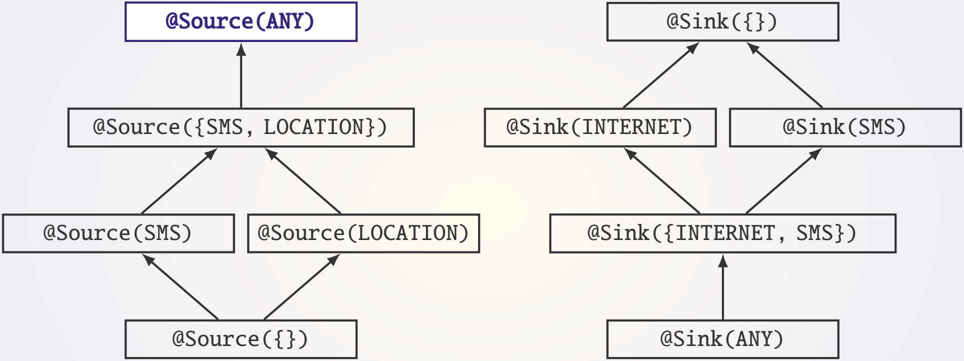
Developer annotations are not trusted

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

Type hierarchy for Sources and Sinks

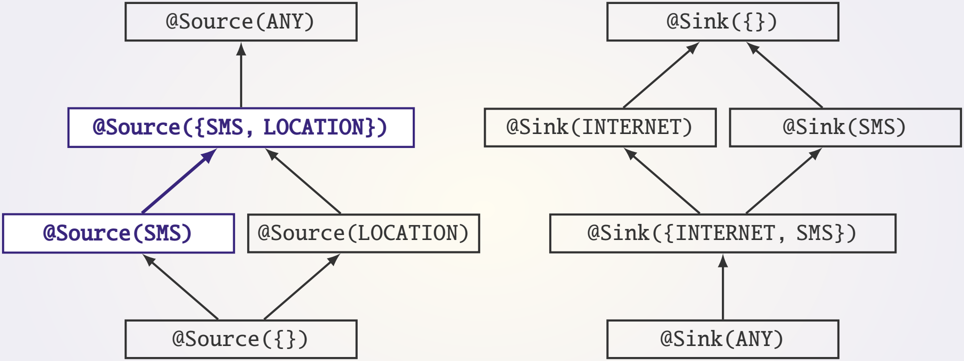


Type hierarchy for Sources and Sinks



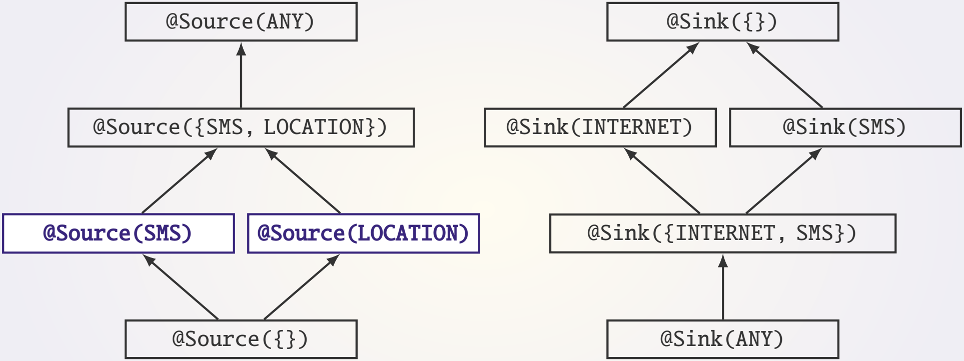
@Source(ANY) ≡ @Source({SMS, LOCATION, INTERNET, ...})

Type hierarchy for Sources and Sinks



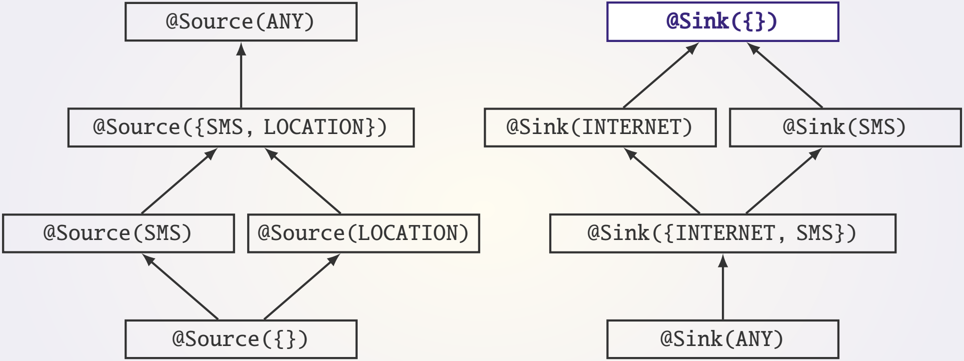

```
@Source(SMS)String sms = ...;
@Source({SMS, LOCATION})String smsLoc = sms;
```

Type hierarchy for Sources and Sinks

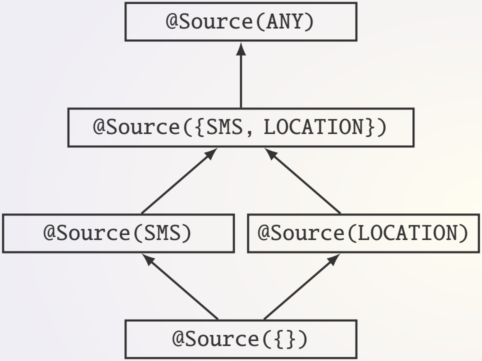


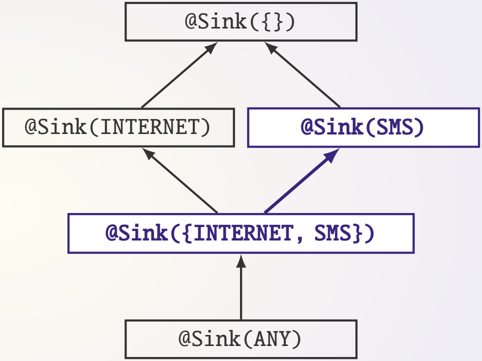

```
@Source(SMS)String sms = ...;
@Source(LOCATION)String loc = sms;
```

Type hierarchy for Sources and Sinks



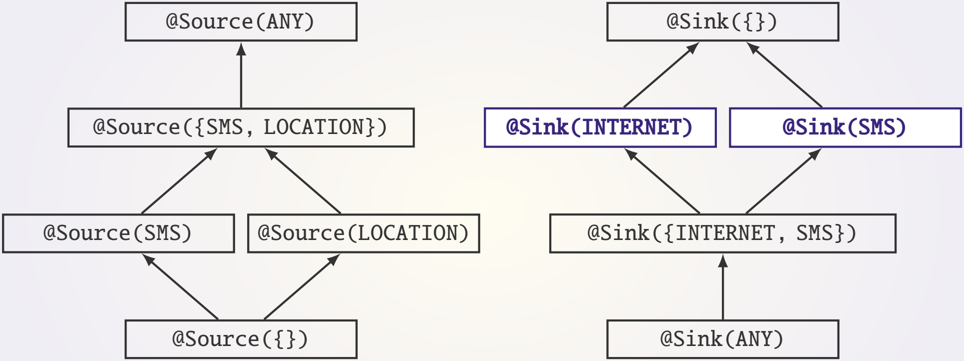
Type hierarchy for Sources and Sinks





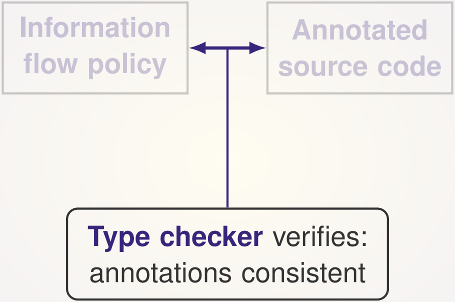

```
@Sink({INTERNET, SMS})String toInetSms;
@Sink(SMS)String toSms = toInetSms;
```

Type hierarchy for Sources and Sinks




```
@Sink(SMS)String toSms;
@Sink(INTERNET)String toInet = toSms;
```

Verification of information flow



Information Flow Type-checker (IFT): Overview

Guarantees of type-checking

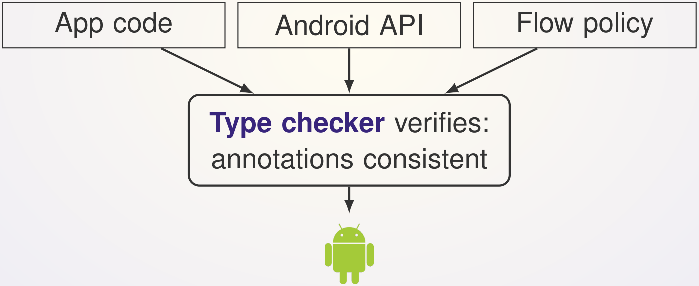
1. Annotations are consistent with code (type correctness)
2. Annotations are consistent with flow policy

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



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

```
@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

```
@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

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

Flow policy

```
LOCATION -> INTERNET
```

Type checker verifies: annotations consistent

Incompatible sinks:
INTERNET $\not<:$ SMS

Information Flow Type-checker (IFT): Example

App code

```
@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

```
@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

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

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

False positives and declassifications

App code

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

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

False positives and declassifications

App code

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

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

@Source(LOCATION)

@Source(LOCATION, SMS)

False positives and declassifications

App code

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

Declassifications

- 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

```
@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

```
@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

App code

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

Implicit information flow

App code

```
@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

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

USER_INPUT -> CONDITIONAL

Our approach: Prune irrelevant conditions

- Add additional Sink CONDITIONAL
- Type-checking warning for conditions with sensitive Source

Implicit information flow

App code

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

USER_INPUT -> CONDITIONAL

Our approach: Prune irrelevant conditions

- Add additional Sink `CONDITIONAL`
- Type-checking warning for conditions with sensitive Source
- 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

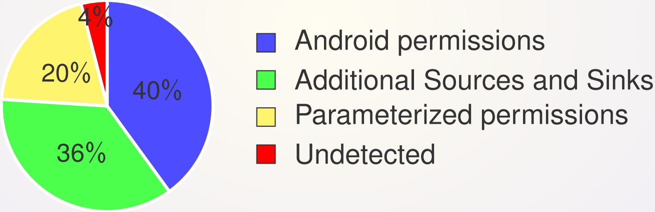
- 5 independent red teams
- 72 Android apps (47 malicious — information-flow malware)
- 8,000 LOC and 12 permissions on average

Adversarial red team challenge

Setup

- 5 independent red teams
- 72 Android apps (47 malicious — information-flow malware)
- 8,000 LOC and 12 permissions 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 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 on average

Results

Usability study

Setup

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

Usability study

Setup

- 2 groups acting as annotators and auditors
- 11 Android apps (1 malicious)
- 900 LOC and 12 permissions 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 cont.

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 cont.

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

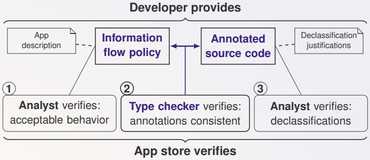
- Crowd-sourcing *(Agarwal & Hall, MobiSys'13)*
- Natural language processing *(Pandita et al., USENIX'13)*
- Clustering *(Gorla et al., ICSE'14)*.

Could aid manual verification of flow policies

Conclusions

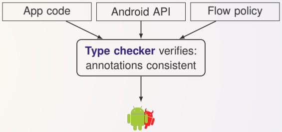
Collaborative verification model

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



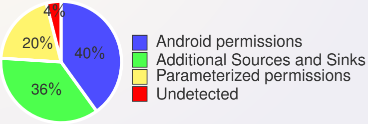
Information Flow Type-checker (IFT)

- 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