Android Taint Flow Analysis for App Sets

Will Klieber*, Lori Flynn,

Amar Bhosale , Limin Jia, and Lujo Bauer

Carnegie Mellon University

*presenting

Motivation

- Detect malicious apps that leak sensitive data.
  - E.g., leak contacts list to marketing company.
  - “All or nothing” permission model.
- Apps can collude to leak data.
  - Evades precise detection if only analyzed individually.
- We build upon **FlowDroid**.
  - FlowDroid alone handles only intra-component flows.
  - We extend it to handle inter-app flows.

Introduction: Android

- Android apps have four types of **components**:
  - Activities **(our focus)**
  - Services
  - Content providers
  - Broadcast receivers
- **Intents** are messages to components.
  - Explicit or implicit designation of recipient
- Components declare **intent filters** to receive implicit intents.
- Matched based on properties of intents, e.g.:
  - Action string (e.g., "`android.intent.action.VIEW`")
  - Data MIME type (e.g., "`image/png`")

Introduction

- **Taint Analysis** tracks the flow of sensitive data.
  - Can be static analysis or dynamic analysis.
  - Our analysis is static.
- We build upon existing Android static analyses:
  - **FlowDroid** [1]: finds intra-component information flow
  - **Epicc** [2]: identifies intent specifications

[1] S. Arzt et al., "FlowDroid: Precise Context, Flow, Field, Object-sensitive and Lifecycle-aware Taint Analysis for Android Apps". *PLDI*, **2014**.

[2] D. Octeau et al., "Effective inter-component communication mapping in Android with Epicc: An essential step towards holistic security analysis". *USENIX Security*, **2013**.

Our Contribution

- We developed a static analyzer called "**DidFail**" ("Droid Intent Data Flow Analysis for Information Leakage").
  - Finds flows of sensitive data across app boundaries.
  - Source code and binaries available at: (or google "DidFail SOAP")
    `http://www.cert.org/secure-coding/tools/didfail.cfm`
- Two-phase analysis:
  1. Analyze each app in isolation.
  2. Use the result of Phase-1 analysis to determine inter-app flows.
- We tested our analyzer on two sets of apps.

Terminology

Definition. A *source* is an <u>external</u> resource (external to the app, not necessarily external to the phone) from which data is read.

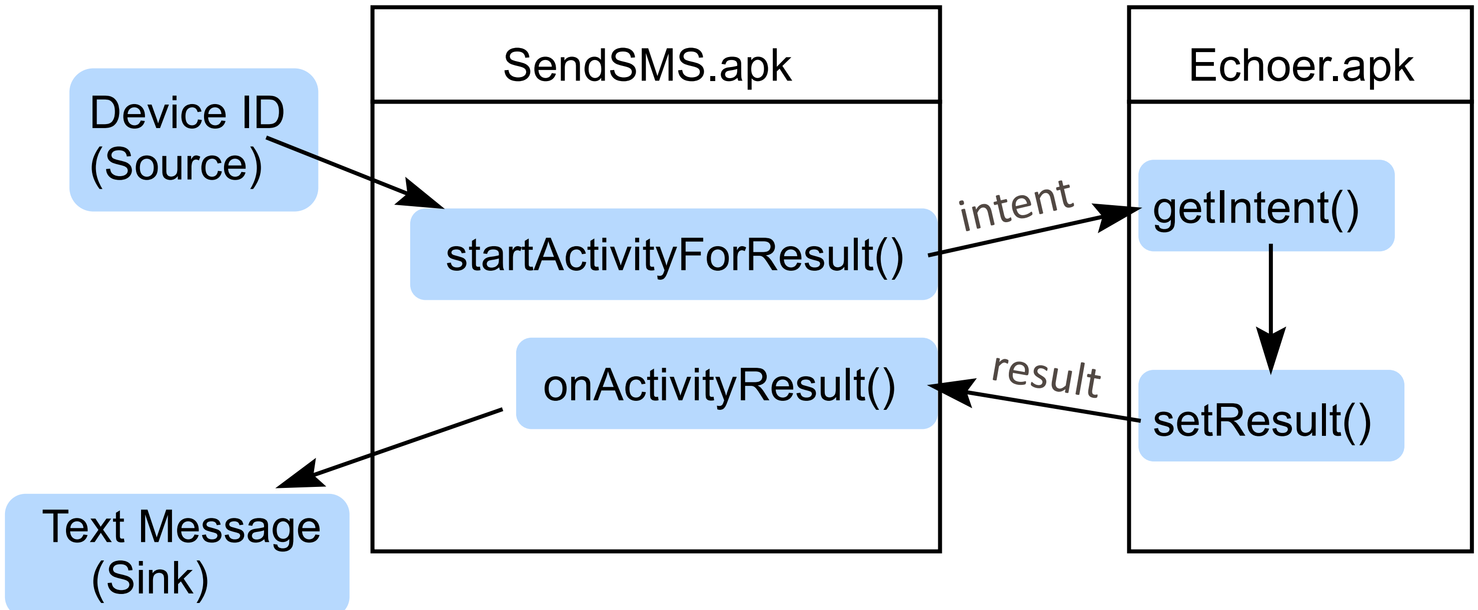
Definition. A *sink* is an <u>external</u> resource to which data is written.

For example,

- **Sources**: Device ID, contacts, photos, current location, etc.
- **Sinks**: Internet, outbound text messages, file system, etc.

Motivating Example

- App *SendSMS.apk* sends an **intent** (a message) to *Echoer.apk*, which sends a **result** back.

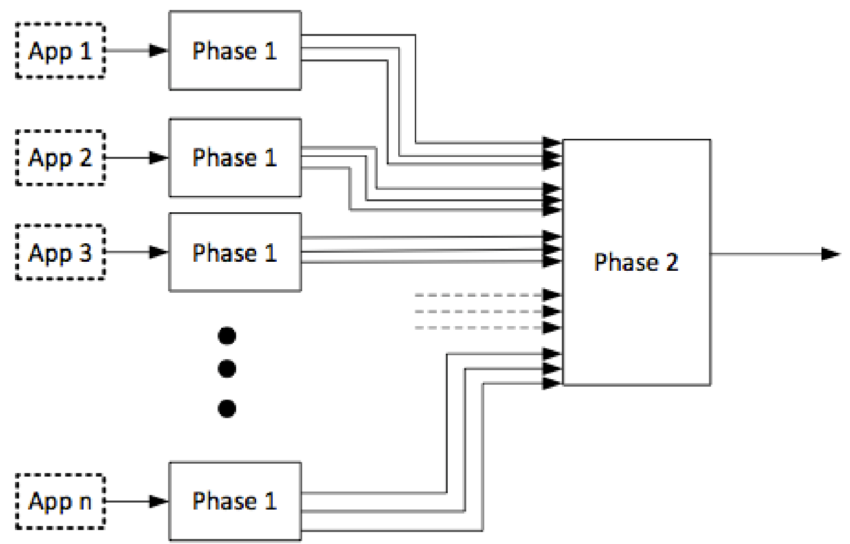


- SendSMS.apk tries to launder the taint through Echoer.apk.
- Existing static analysis tools cannot precisely detect such inter-app data flows.

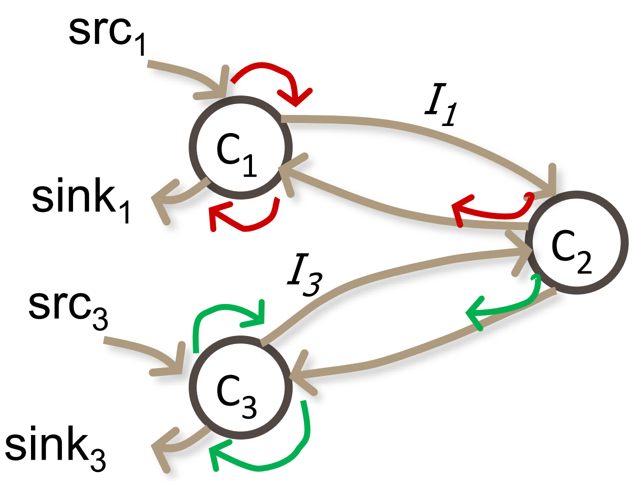
Analysis Design

- **Phase 1**: Each app analyzed once, in isolation.
  - **FlowDroid:** Finds tainted dataflow from sources to sinks.
    - Received intents are considered sources.
    - Sent intent are considered sinks.
  - **Epicc:** Determines properties of intents.
  - Each intent-sending call site is labelled with a unique *intent ID*.

- **Phase 2**: Analyze a set of apps:
  - For each intent **sent** by a component, determine which components can **receive** the intent.
  - Generate & solve taint flow equations.



Running Example



Three components: C_1, C_2, C_3.

C1 = SendSMS

C2 = Echoer

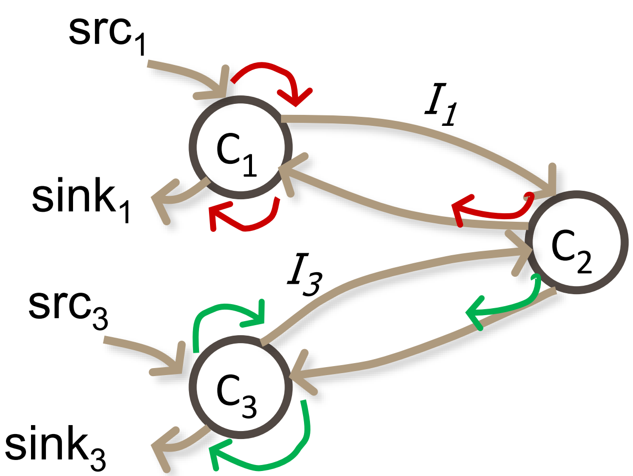
C3 is similar to C1

For $i \in \{1, 3\}$:

- C_i sends data from src_i to component C_2 via intent I_i.
- C_2 reads data from intent I_i and echoes it back to C_i.
- C_i reads data from the result and writes it to $sink_i$.

- $sink_1$ is tainted with only src_1.
- $sink_3$ is tainted with only src_3.

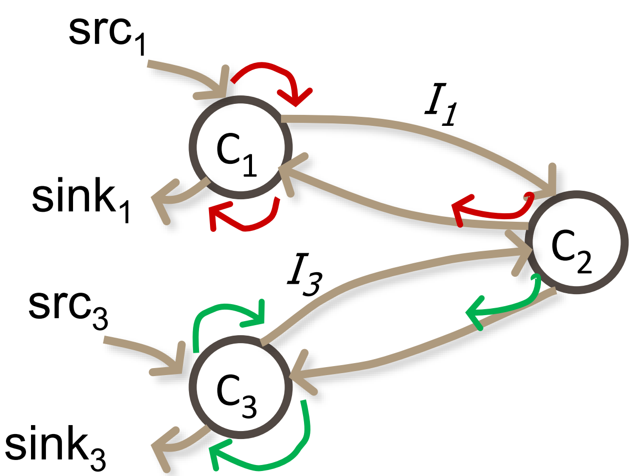
Running Example



Notation:

- "$src \xrightarrow{C} sink$": Flow from src to $sink$ in C.
- "$I(C_{\mathrm{TX}}, C_{\mathrm{RX}}, id)$": Intent from C_{TX} to C_{RX} with ID id.
- "$R(I)$": Response (result) for intent I.
- "$T(s)$": Set of sources with which s is tainted.

Running Example



$$src_1 \xrightarrow{C_1} I(C_1, C_2, id_1)$$

$$I(C_1, C_2, id_1) \xrightarrow{C_2} R(I(C_1, C_2, id_1))$$

$$R(I(C_1, C_2, id_1)) \xrightarrow{C_1} sink_1$$

$$src_3 \xrightarrow{C_3} I(C_3, C_2, id_3)$$

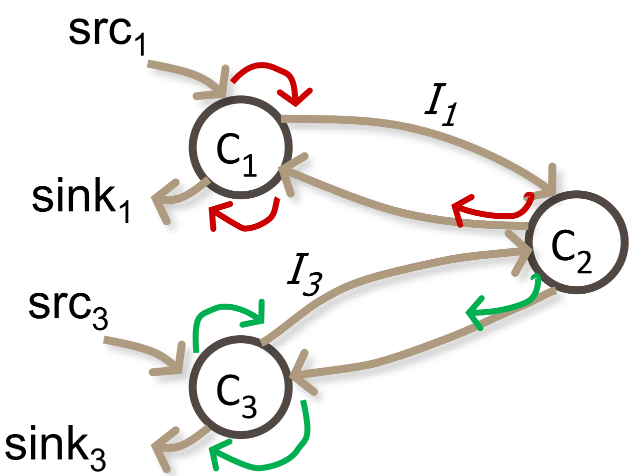
$$I(C_3, C_2, id_3) \xrightarrow{C_2} R(I(C_3, C_2, id_3))$$

$$R(I(C_3, C_2, id_3)) \xrightarrow{C_3} sink_3$$

Notation:

- "$src \xrightarrow{C} sink$": Flow from src to $sink$ in C.
- "$I(C_{\text{TX}}, C_{\text{RX}}, id)$": Intent from C_{TX} to C_{RX} with ID id.
- "$R(I)$": Response (result) for intent I.
- "$T(s)$": Set of sources with which s is tainted.

Running Example



$$src_1 \xrightarrow{C_1} I(C_1, C_2, id_1)$$
$$I(C_1, C_2, id_1) \xrightarrow{C_2} R(I(C_1, C_2, id_1))$$
$$R(I(C_1, C_2, id_1)) \xrightarrow{C_1} sink_1$$

$$src_3 \xrightarrow{C_3} I(C_3, C_2, id_3)$$
$$I(C_3, C_2, id_3) \xrightarrow{C_2} R(I(C_3, C_2, id_3))$$
$$R(I(C_3, C_2, id_3)) \xrightarrow{C_3} sink_3$$

Final Sink Taints:

- $T(sink_1) = \{src_1\}$
- $T(sink_3) = \{src_3\}$

Notation:

- "$src \xrightarrow{C} sink$": Flow from *src* to *sink* in C.
- "$I(C_{\mathrm{TX}}, C_{\mathrm{RX}}, id)$": Intent from C_{TX} to C_{RX} with ID id.
- "$R(I)$": Response (result) for intent I.
- "$T(s)$": Set of sources with which s is tainted.

Phase1 Flow Equations

Analyze each component separately.

Phase 1 Flow Equations:

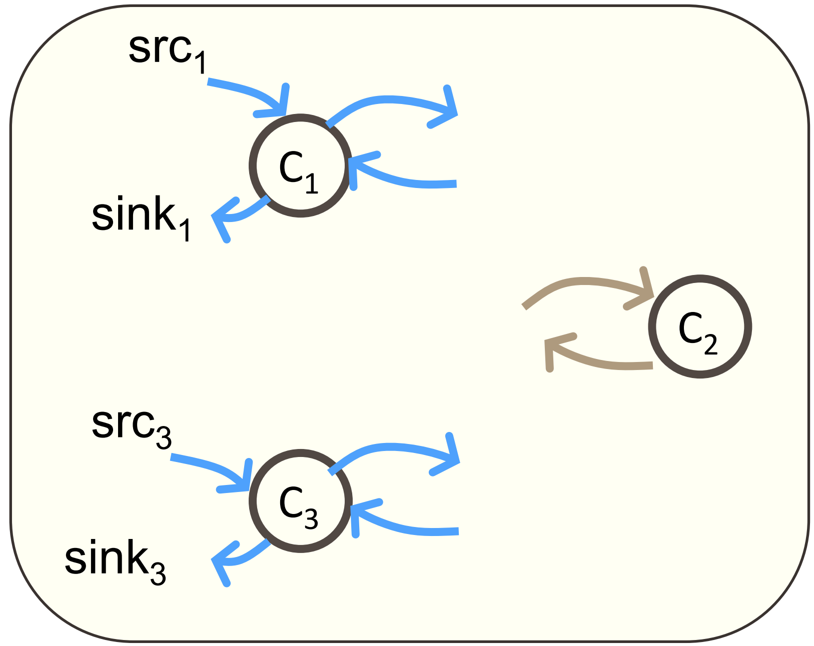
$$src_1 \xrightarrow{C_1} I(C_1, *, id_1)$$

$$R(I(C_1, *, *)) \xrightarrow{C_1} sink_1$$

$$I(*, C_2, *) \xrightarrow{C_2} R(I(*, C_2, *))$$

$$src_3 \xrightarrow{C_3} I(C_3, *, id_3)$$

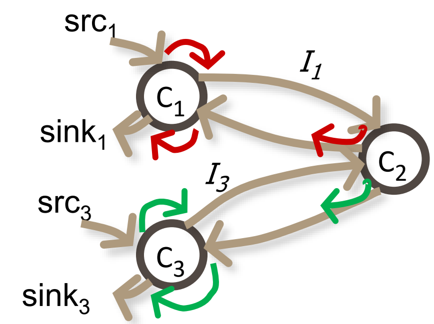
$$R(I(C_3, *, *)) \xrightarrow{C_3} sink_3$$



Notation

- "$src \xrightarrow{C} sink$": Flow from *src* to *sink* in C.
- "$I(C_{\mathrm{TX}}, C_{\mathrm{RX}}, id)$": Intent from C_{TX} to C_{RX} with ID *id*.
- "$R(I)$": Response (result) for intent I.
- An asterisk ("$*$") indicates an unknown component.

Phase2 Flow Equations



Instantiate Phase-1 equations for all possible sender/receiver pairs.

Phase 1 Flow Equations:

$$src_1 \xrightarrow{C_1} I(C_1, *, id_1)$$

$$R(I(C_1, *, *)) \xrightarrow{C_1} sink_1$$

$$I(*, C_2, *) \xrightarrow{C_2} R(I(*, C_2, *))$$

$$src_3 \xrightarrow{C_3} I(C_3, *, id_3)$$

$$R(I(C_3, *, *)) \xrightarrow{C_3} sink_3$$

Phase 2 Flow Equations:

$$src_1 \xrightarrow{C_1} I(C_1, C_2, id_1)$$

$$R(I(C_1, C_2, id_1)) \xrightarrow{C_1} sink_1$$

$$I(C_1, C_2, id_1) \xrightarrow{C_2} R(I(C_1, C_2, id_1))$$

$$I(C_3, C_2, id_3) \xrightarrow{C_2} R(I(C_3, C_2, id_3))$$

$$src_3 \xrightarrow{C_3} I(C_3, C_2, id_3)$$

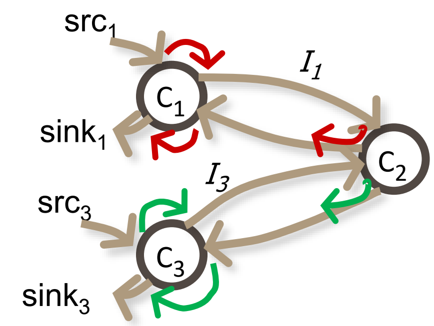
$$R(I(C_3, C_2, id_3)) \xrightarrow{C_3} sink_3$$

Notation

- "$src \xrightarrow{C} sink$": Flow from src to $sink$ in C.
- "$I(C_{\mathrm{TX}}, C_{\mathrm{RX}}, id)$": Intent from C_{TX} to C_{RX} with ID id.
- "$R(I)$": Response (result) for intent I.

Phase 2 Taint Equations

For each flow equation "src → sink",
generate taint equation "T(src) ⊆ T(sink)".



Phase 2 Flow Equations:

$$src_1 \xrightarrow{C_1} I(C_1, C_2, id_1)$$
$$R(I(C_1, C_2, id_1)) \xrightarrow{C_1} sink_1$$
$$I(C_1, C_2, id_1) \xrightarrow{C_2} R(I(C_1, C_2, id_1))$$
$$I(C_3, C_2, id_3) \xrightarrow{C_2} R(I(C_3, C_2, id_3))$$
$$src_3 \xrightarrow{C_3} I(C_3, C_2, id_3)$$
$$R(I(C_3, C_2, id_3)) \xrightarrow{C_3} sink_3$$

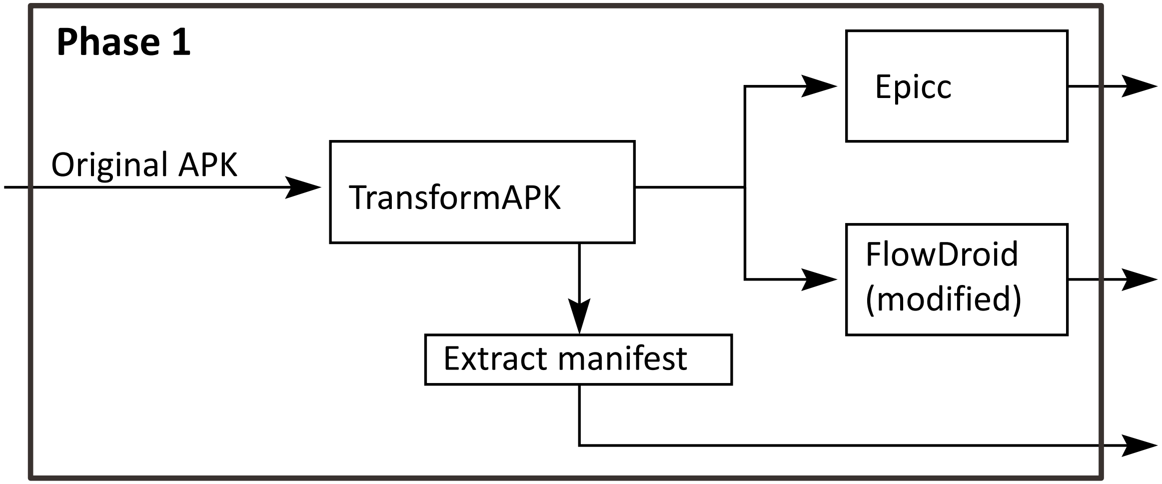
Phase 2 Taint Equations:

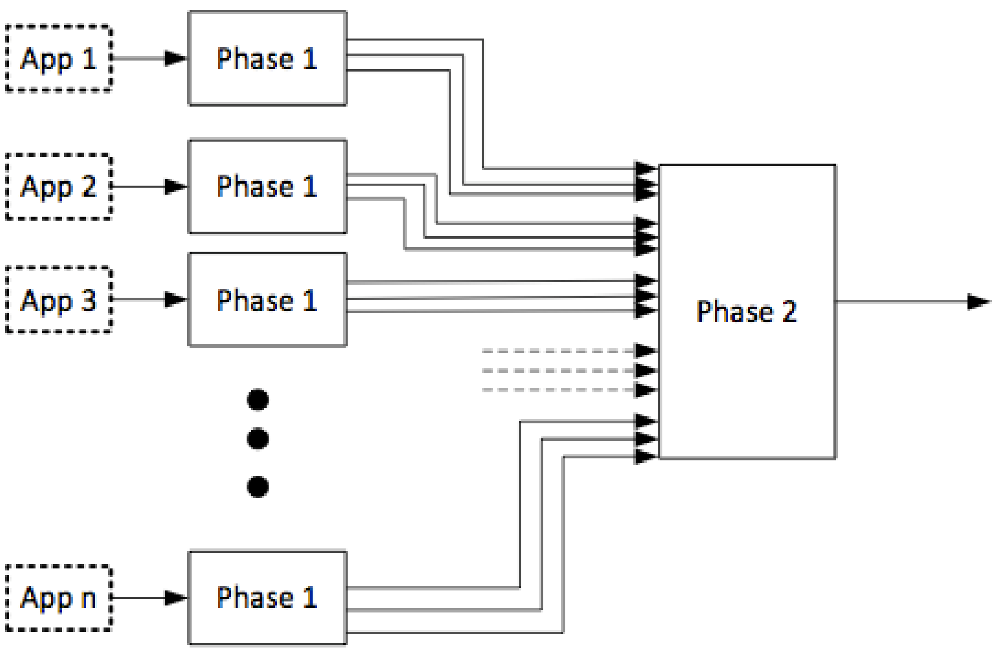
$$T(src_1) \subseteq T(I(C_1, C_2, id_1))$$
$$T(R(I(C_1, C_2, id_1))) \subseteq T(sink_1)$$
$$T(I(C_1, C_2, id_1)) \subseteq T(R(I(C_1, C_2, id_1)))$$
$$T(I(C_3, C_2, id_1)) \subseteq T(R(I(C_3, C_2, id_3)))$$
$$T(src_3) \subseteq T(I(C_3, C_2, id_3))$$
$$T(R(I(C_3, C_2, id_3))) \subseteq T(sink_3)$$

Notation

- "$src \xrightarrow{C} sink$": Flow from *src* to *sink* in C.
- "$I(C_{\mathrm{TX}}, C_{\mathrm{RX}}, id)$": Intent from C_{TX} to C_{RX} with ID *id*.
- "$R(I)$": Response (result) for intent I.
- "$T(s)$": Set of sources with which s is tainted.

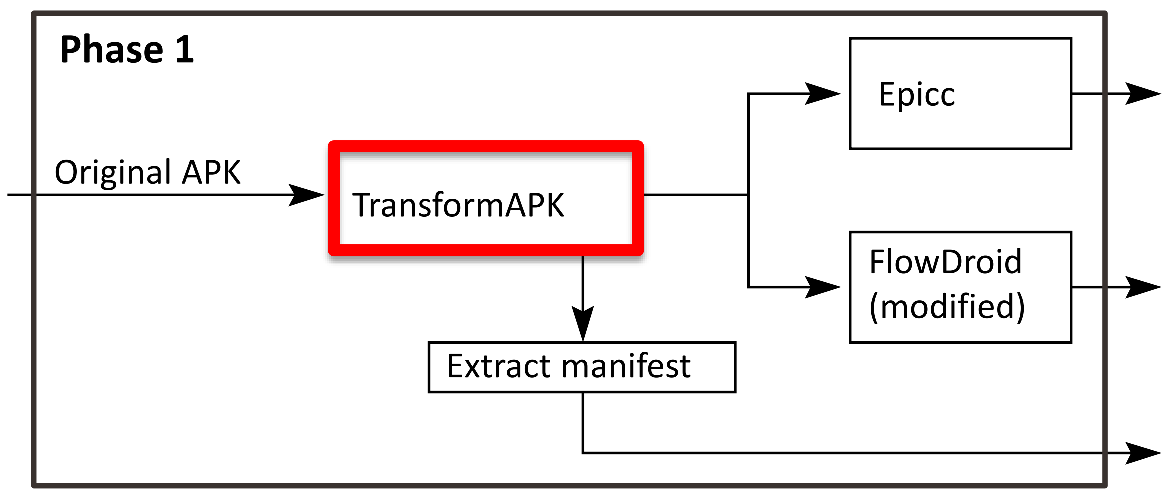
If *s* is a non-intent source, then T(*s*) = {*s*}.


Phase 1
Original APK
TransformAPK
Epicc
FlowDroid
(modified)
Extract manifest


App 1
App 2
App 3
App n
Phase 1
Phase 1
Phase 1
Phase 1
Phase 2

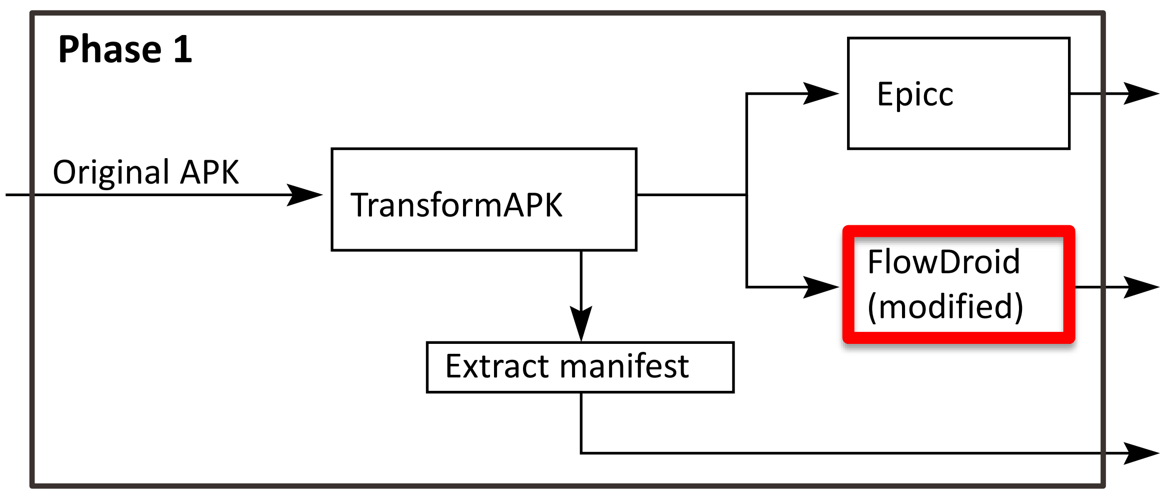
Implementation: Phase 1

- **APK Transformer**
  - Assigns unique Intent ID to each call site of intent-sending methods.
    - Enables matching intents from the output of FlowDroid and Epicc
  - Uses Soot to read APK, modify code (in Jimple), and write new APK.
  - Problem: Epicc is closed-source. How to make it emit Intent IDs?
  - Solution (hack): Add `putExtra` call with Intent ID.



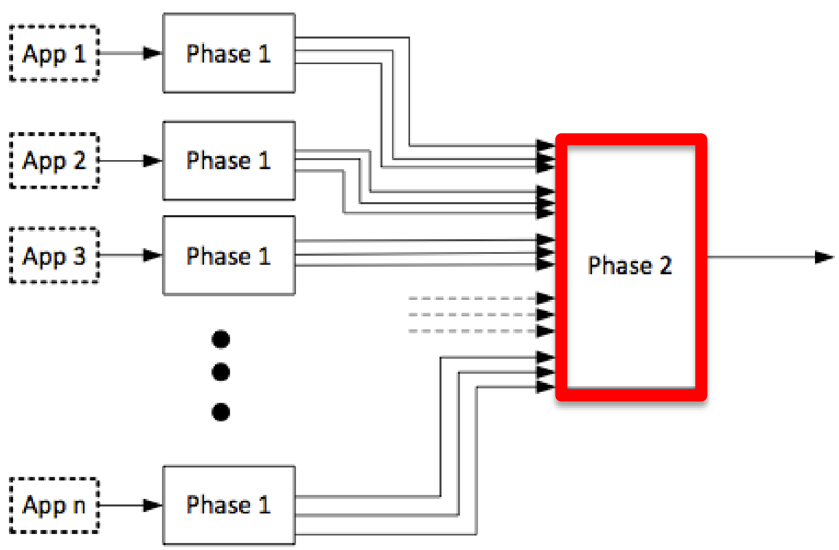
Implementation: Phase 1

- **FlowDroid Modifications:**
  - Extract intent IDs inserted by APK Transformer, and include in output.
  - When sink is an intent, identify the sending component.
    - In `base.startActivity`, assume base is the sending component. (Soundness?)
  - For deterministic output: Sort the final list of flows.



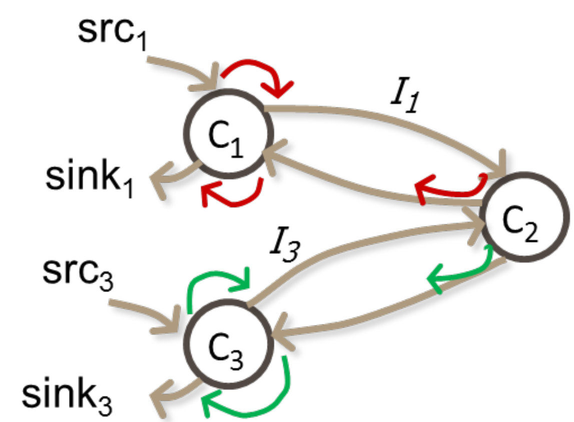
Implementation: Phase 2

- **Phase 2**
  - Take the Phase 1 output.
  - Generate and solve the data-flow equations.
  - Output:
    1. Directed graph indicating information flow between sources, intents, intent results, and sinks.
    2. Taintedness of each sink.



Testing DidFail analyzer: App Set 1

- **SendSMS.apk**
  - Reads device ID, passes through Echoer, and leaks it via SMS
- **Echoer.apk**
  - Echoes the data received via an intent
- **WriteFile.apk**
  - Reads physical location (from GPS), passes through Echoer, and writes it to a file



- $getDeviceId \xrightarrow{SendSMS} startActivityForResult$

  $getIntent \xrightarrow{Echoer} setResult$

  $onActivityResult \xrightarrow{SendSMS} sendTextMessage$

- $getLastKnownLocation \xrightarrow{WriteFile} startActivityForResult$

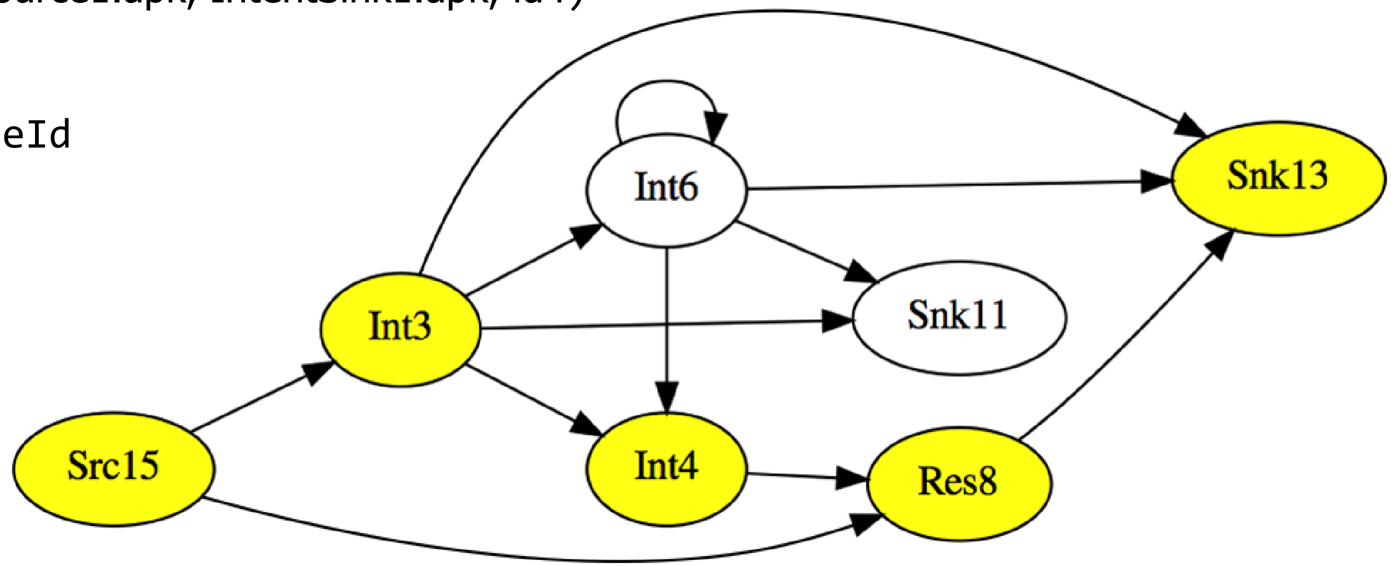
  $getIntent \xrightarrow{Echoer} setResult$

  $onActivityResult \xrightarrow{WriteFile} write$

Testing DidFail analyzer: App Set 2 (DroidBench)

```
Int3  = I(IntentSink2.apk, IntentSource1.apk, id3)
Int4  = I(IntentSource1.apk, IntentSink1.apk, id4)
Res8  = R(Int4)
Src15 = getDeviceId
Snk13 = Log.i
```

Graph generated using GraphViz.



Some taint flows:

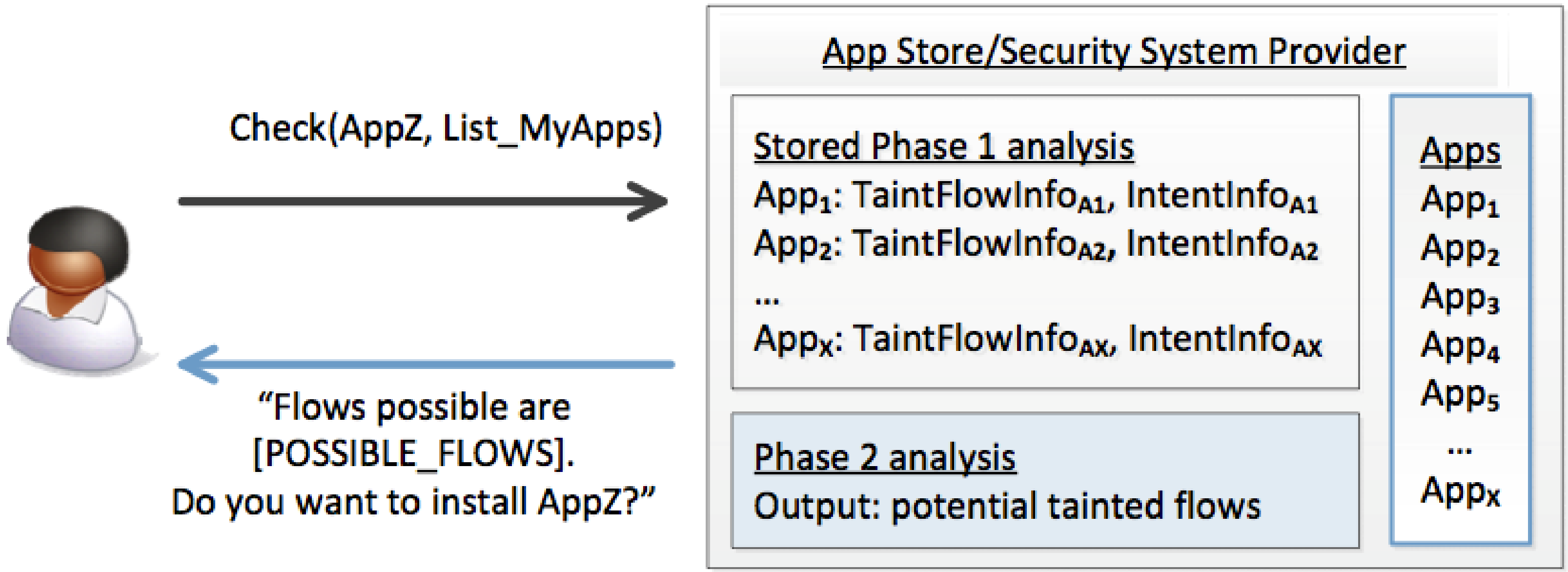
- $Src15 \xrightarrow{IntentSink2} Int3 \xrightarrow{IntentSource1} Snk13$
- $Src15 \xrightarrow{IntentSink2} Int3 \xrightarrow{IntentSource1} Int4 \xrightarrow{IntentSink1} Res8 \xrightarrow{IntentSource1} Snk13$
- $Src15 \xrightarrow{IntentSink1} Res8 \xrightarrow{IntentSource1} Snk13$

Limitations

- **Unsoundness**
  - Inherited from FlowDroid/Epicc
    - Native code, reflection, etc.
  - Shared static fields
  - Implicit flows
  - Currently, only activity intents
  - Bugs
- **Imprecision**
  - Inherited from FlowDroid/Epicc
  - DidFail doesn’t consider permissions when matching intents
  - All intents received by a component are conflated together as a single source

Use of Two-Phase Approach in App Stores

- We envision that the two-phase analysis can be used as follows:
  - An app store runs the phase-1 analysis for each app it has.
  - When the user wants to download a new app, the store runs the phase-2 analysis and indicates new flows.
  - Fast response to user.



DidFail vs IccTA

- IccTA was developed (at roughly the same time as DidFail) by:
  - Li Li, Alexandre Bartel, Jacques Klein, Yves Le Traon (Luxembourg);
  - Steven Arzt, Siegfried Rasthofer, Eric Bodden (EC SPRIDE);
  - Damien Octeau, Patrick McDaniel (Penn State).
- IccTA uses a one-phase analysis
  - IccTA is more precise than DidFail's two-phase analysis.
  - Two-phase DidFail analysis allows fast 2nd-phase computation.
- Future collaboration between IccTA and DidFail teams?

Conclusion

- We introduced a new analysis that integrates and enhances existing Android app static analyses.
- Demonstrated feasibility by implementing a prototype and testing it.
- Two-phase analysis can be used by app store to provide fast response.
- Future work:
  - Implicit flows
  - Static fields
  - Distinguish different received intents
  - Other data channels (file system, non-activity intents)
  - Etc.

Thank You