WILLIAM & MARY

CHARTERED 1693

CSCI 445: Mobile Application Security

Lecture 19

Prof. Adwait Nadkarni

Announcements

1. Application Analysis (Milestone 4) *minimum requirements*:
   - Team of 1: 3 RQs, >=1 from each research goal.
   - Team of 2: 5 RQs, >=1 from each research goal.
2. **Final report: due on 05/02,** *extensions on a case-by-case basis*
3. **Project Presentations** (04/30): ***up to 5 bonus points***
   - RQs, progress, problems/challenges, anticipated results
   - (approx.) 7-8 minute duration + 2 minutes for questions (depending on how many groups present)
     - ***Let me know by 04/18*** *if you want to present.*

How do we study apps?

- Generally, two ways to do this:
- *Static analysis* tells you *what can potentially happen*
  - Getting source code: ded, dex2jar, jadx, androguard
  - Extending existing analysis frameworks (e.g., Fortify, soot)
  - Frameworks targeted at Android: FlowDroid, Argus
- *Dynamic analysis* tells you *what actually happened* in a specific runtime environment
  - Several tools: TaintDroid, DroidScope
  - Derivative environments: Droidbox, andrubis, MarvinSafe
  - *Hard to automate*; need to explore every code path in the app

Soundness vs Precision

- When analyzing applications,
- **Sound analysis:** Detects every instance of target/bad behavior, i.e., doesn’t miss anything (i.e., *no false negatives*)
- **Precise analysis:** Detects only true instances of target/bad behavior as bad behavior, i.e., doesn’t flag benign things (i.e., *no false positives*)
- Which is sound? Static, or dynamic?
  - Static, *in theory;* soundy in practice
- Which is precise? Static, or dynamic?
  - Dynamic, however, it depends on the granularity

```
Method method = foo.getClass().getMethod("doSomethingEvil", null);
method.invoke(foo, null);
```

Sound*i*ness

Soundiness Manifesto

- Tools make decisions that sacrifice soundness. Why?
  - Precision, i.e., to reduce FPs
  - Performance(i.e., execution time)
- However, soundy tools are *practical*. So what is the problem?
- **Problem:** Soundness is *assumed* of static analysis tools
  - Unsound choices are *only* known to very few experts

https://www.mobusinc.com/blog/beware-experts-confuse-conceal

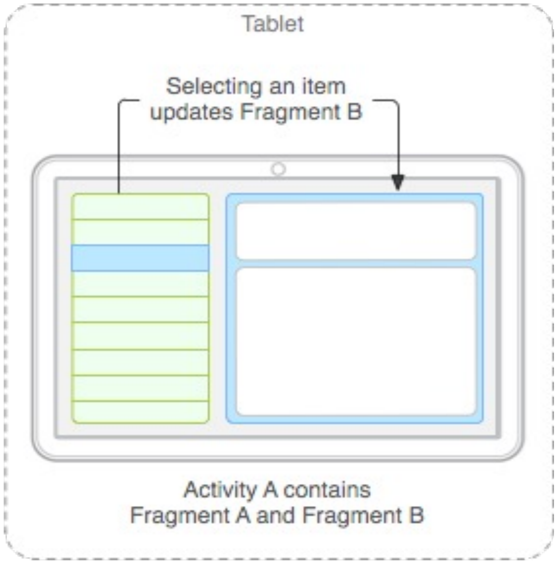
[1] Livshits, Benjamin, Manu Sridharan, Yannis Smaragdakis, Ondřej Lhoták, J. Nelson Amaral, Bor-Yuh Evan Chang, Samuel Z. Guyer, Uday P. Khedker, Anders Møller, and Dimitrios Vardoulakis. "In defense of soundiness: a manifesto." *Communications of the ACM* 58, no. 2 (2015): 44-46.

Motivation

- The soundiness manifesto talks about unsound choices in terms of *unsupported language features* (e.g., reflection, JNI)
- Unsound choices need to be made explicit. Why?
    1. We (analysts, researchers) need to know the limitations of our analysis
    2. These choices *propagate*: Tools that inherit other tools, also inherit their limitations, *sometimes unknowingly*

It's just a bunch of language features. Can't we simply enumerate them and document what a specific tool covers?

Motivating Example

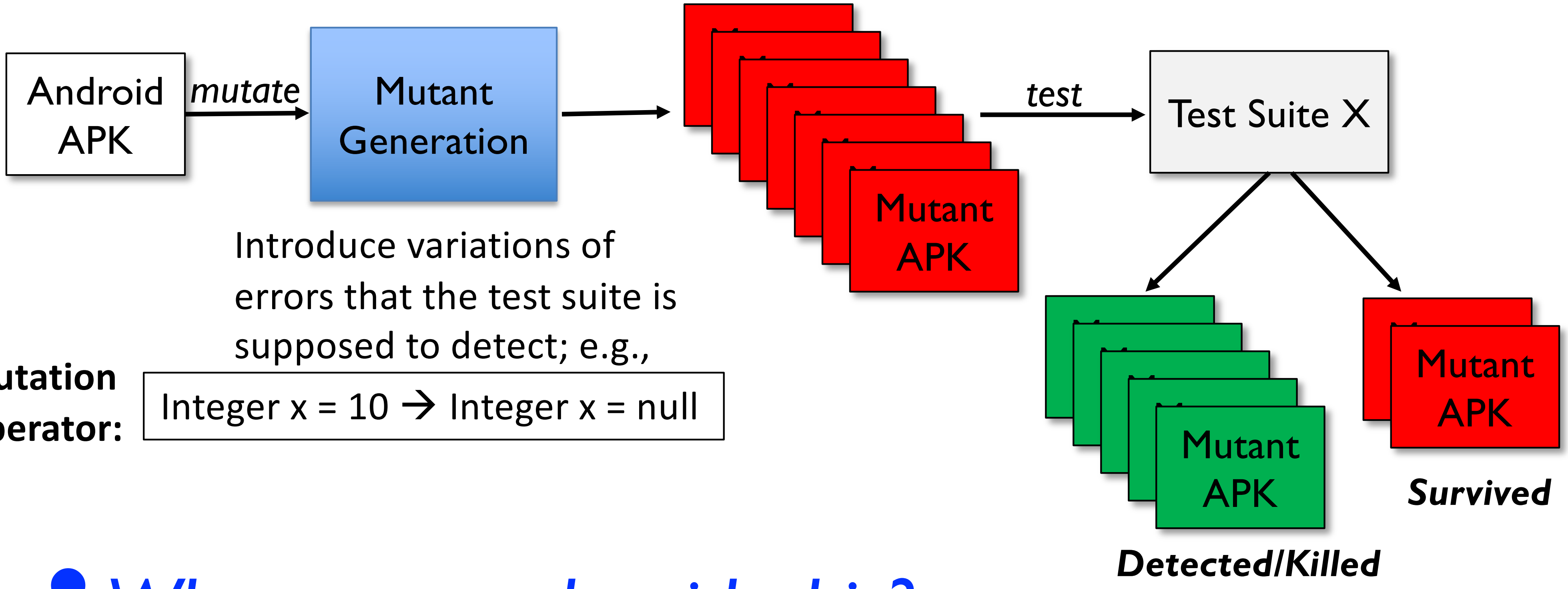

Tablet
Selecting an item updates Fragment B
Activity A contains Fragment A and Fragment B

- *FlowDroid*: Detects data leaks in Android apps
- Preliminary *manual* investigation:
- **Key Finding 1**: FlowDroid v1.0 does not track code inside *fragments*
- *It's not just language features, is it?*
- Reported the flaw, developers fixed it in FlowDroid v2.0
- **Key Finding 2**: We make slight variation in initializing the fragment, and the flaw persists
- **Key Finding 3**: Of the 13 tools that inherit FlowDroid, *only 1 considers this flaw, i.e., flaws propagate!* Often unknowingly.

We need a scalable and efficient technique to systematically detect such unsound choices

Mutation Testing

- Objective (Software Testing): For evaluating the *effectiveness* of test cases/suites



Android APK

mutate

Mutant Generation

Mutant APK

test

Test Suite X

Mutant APK

Detected/Killed

Mutant APK

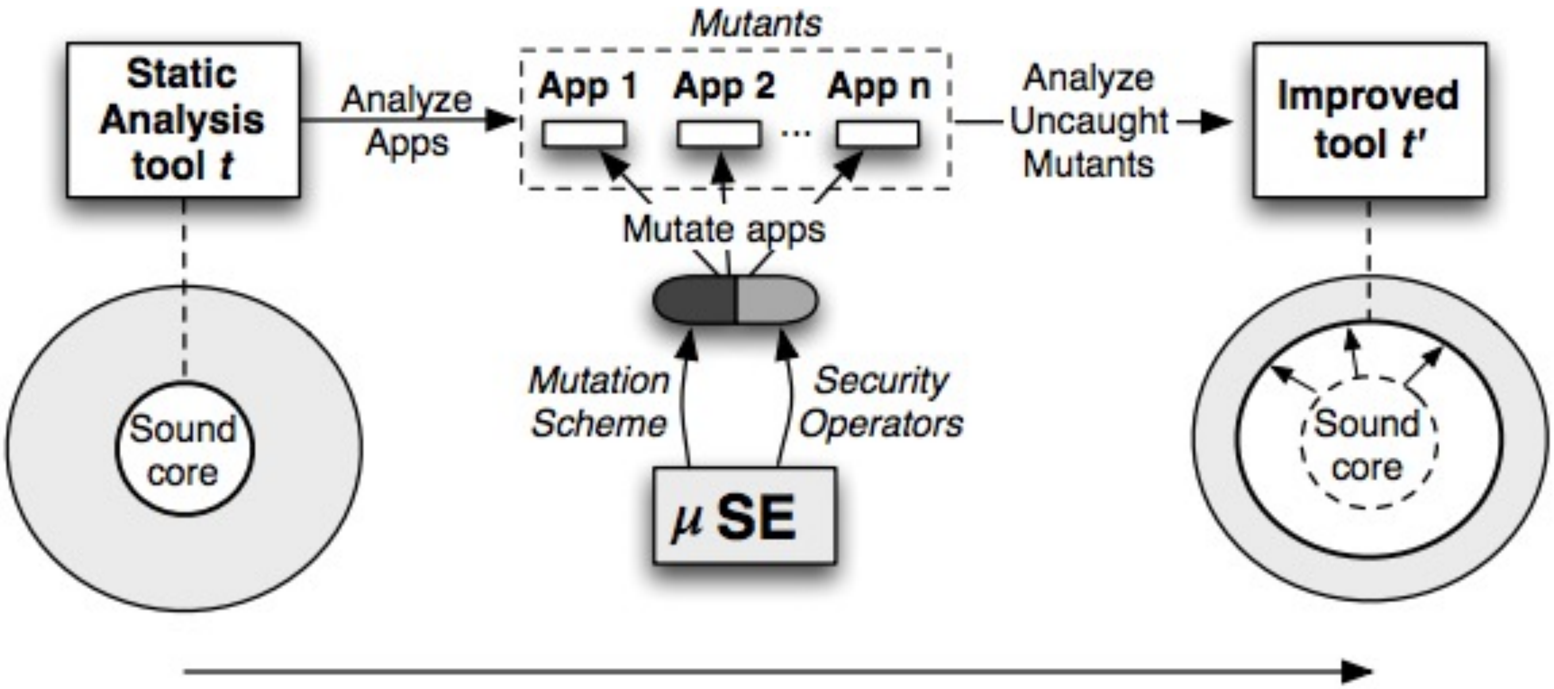
Survived

Introduce variations of errors that the test suite is supposed to detect; e.g.,

Mutation Operator: Integer x = 10 → Integer x = null

- *What can we do with this?*

Mutation-based Soundness Evaluation (mSE)

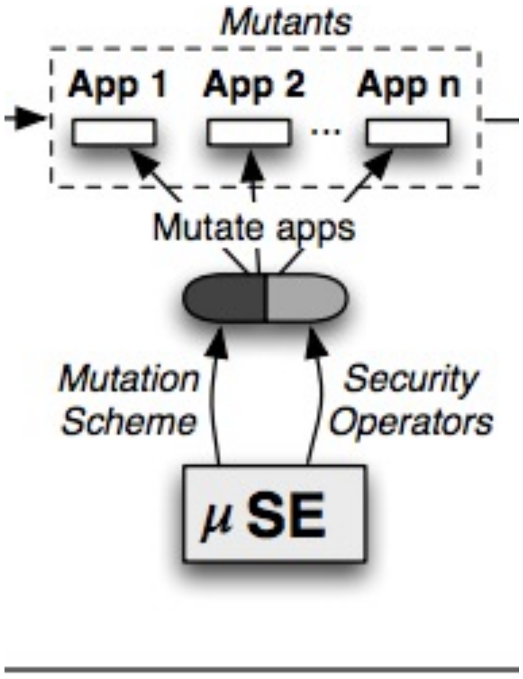


For evaluating Android security tools

1. Mutate apps using *security operators* and *mutation schemes*
2. Run analysis tool on mutants
3. Analyze uncaught mutants to discover unsound decisions

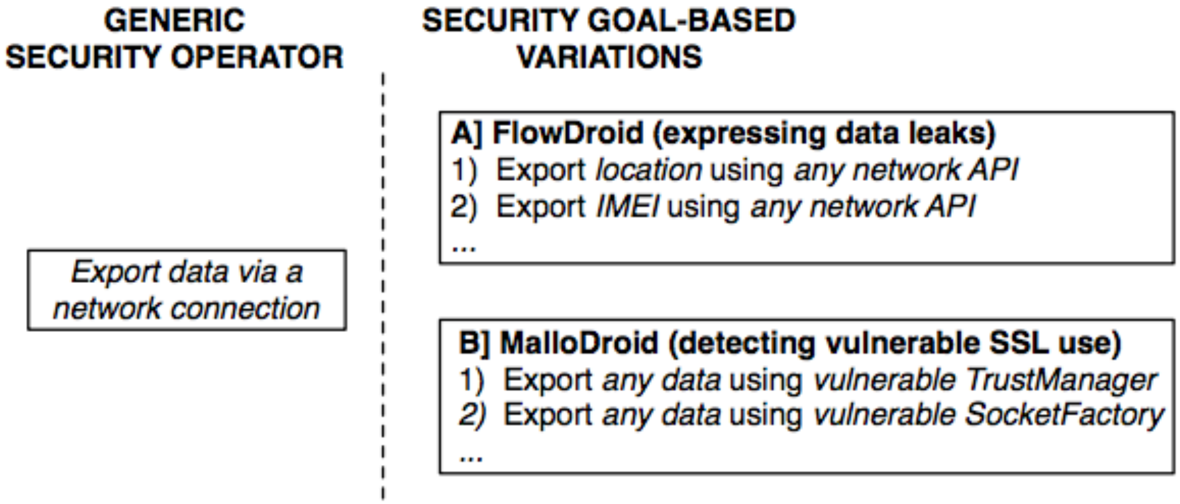
Challenges

- *What problems do we want to express?*
  - SE has mutation operators (i.e., simple code transformations)
  - Mutation operators mimic common software bugs
  - What do we do for security?
- *Where to seed the mutant?*
  - In SE, the general practice is to do it everywhere (especially when adding code).
  - What else can we do for security?



Design: *Security Operators*

- Option A: Tool-specific operators? 100s of tools, *not scalable*
- Option B: *Generic* operators? *Cannot apply to all tools*



GENERIC SECURITY OPERATOR

Export data via a network connection

SECURITY GOAL-BASED VARIATIONS

A] FlowDroid (expressing data leaks)

1) Export *location* using *any network API*
2) Export *IMEI* using *any network API*

...

B] MalloDroid (detecting vulnerable SSL use)

1) Export *any data* using *vulnerable TrustManager*
2) Export *any data* using *vulnerable SocketFactory*

...

- *Security operators* are *bound to the security goal of the analysis (e.g., detecting data leaks, detecting SSL vulnerabilities)*
  - **One-time effort:** A single operator can evaluate a large set of tools (e.g., all tools that detect data leaks, such as FlowDroid, ARGUS, BlueSeal, etc.)

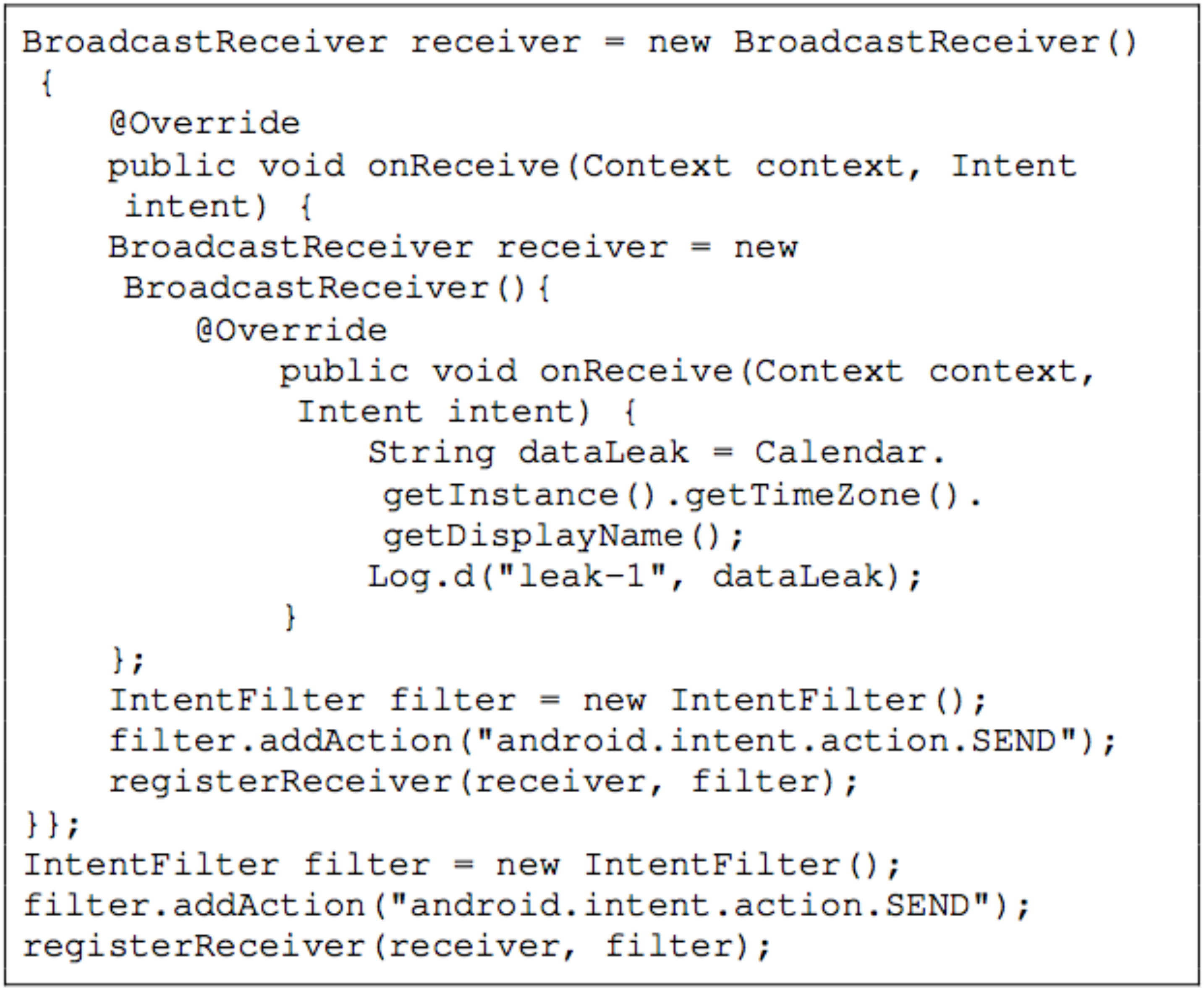
Design: *Mutation Schemes*

- Can we just seed mutants everywhere? Yes, and that's *one* possible strategy.
- Major considerations for **Android** security:

1. Android's *unique abstractions*
    - Activity, fragment, and other component lifecycles
    - Dynamically created callbacks (e.g., dynamically created broadcast receivers, UI callbacks (e.g., onClick() and other callbacks defined in the XML resources)
    - ...

Design: *Mutation Schemes*

- Can we
  one po
- Major
  1. Ar



```
BroadcastReceiver receiver = new BroadcastReceiver()
 {
    @Override
    public void onReceive(Context context, Intent
     intent) {
    BroadcastReceiver receiver = new
     BroadcastReceiver(){
        @Override
            public void onReceive(Context context,
             Intent intent) {
                String dataLeak = Calendar.
                 getInstance().getTimeZone().
                 getDisplayName();
                Log.d("leak-1", dataLeak);
            }
    };
    IntentFilter filter = new IntentFilter();
    filter.addAction("android.intent.action.SEND");
    registerReceiver(receiver, filter);
}};
IntentFilter filter = new IntentFilter();
filter.addAction("android.intent.action.SEND");
registerReceiver(receiver, filter);
```

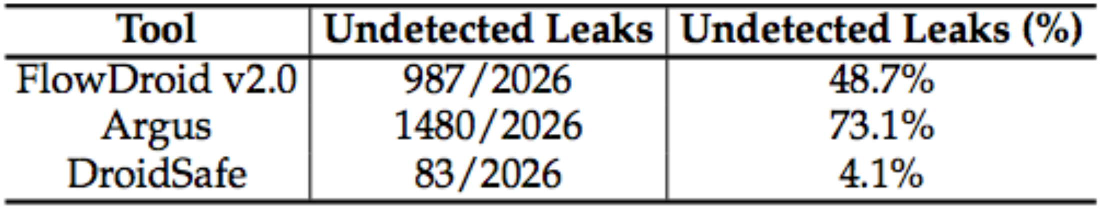
Design: *Mutation Schemes*

- Can we just seed mutants everywhere? Yes, and that's *one* possible strategy.
- Major considerations for Android **security**:

2. Leveraging the *security goal* (e.g., finding data leaks)
    - *Taint-based operator placement:*
        - *Source* in one callback, *sink* in another. E.g., get location in `onStart()` and export `onPause()`
    - *Complex paths:* Make the path between *source* and *sink* as complex as possible (e.g., add lots of function calls in between)

Evaluation

- Data leak detectors: FlowDroid, Argus, DroidSafe
- Create thousands of mutants/leaks using mSE, and then execute analysis tools on the mutants

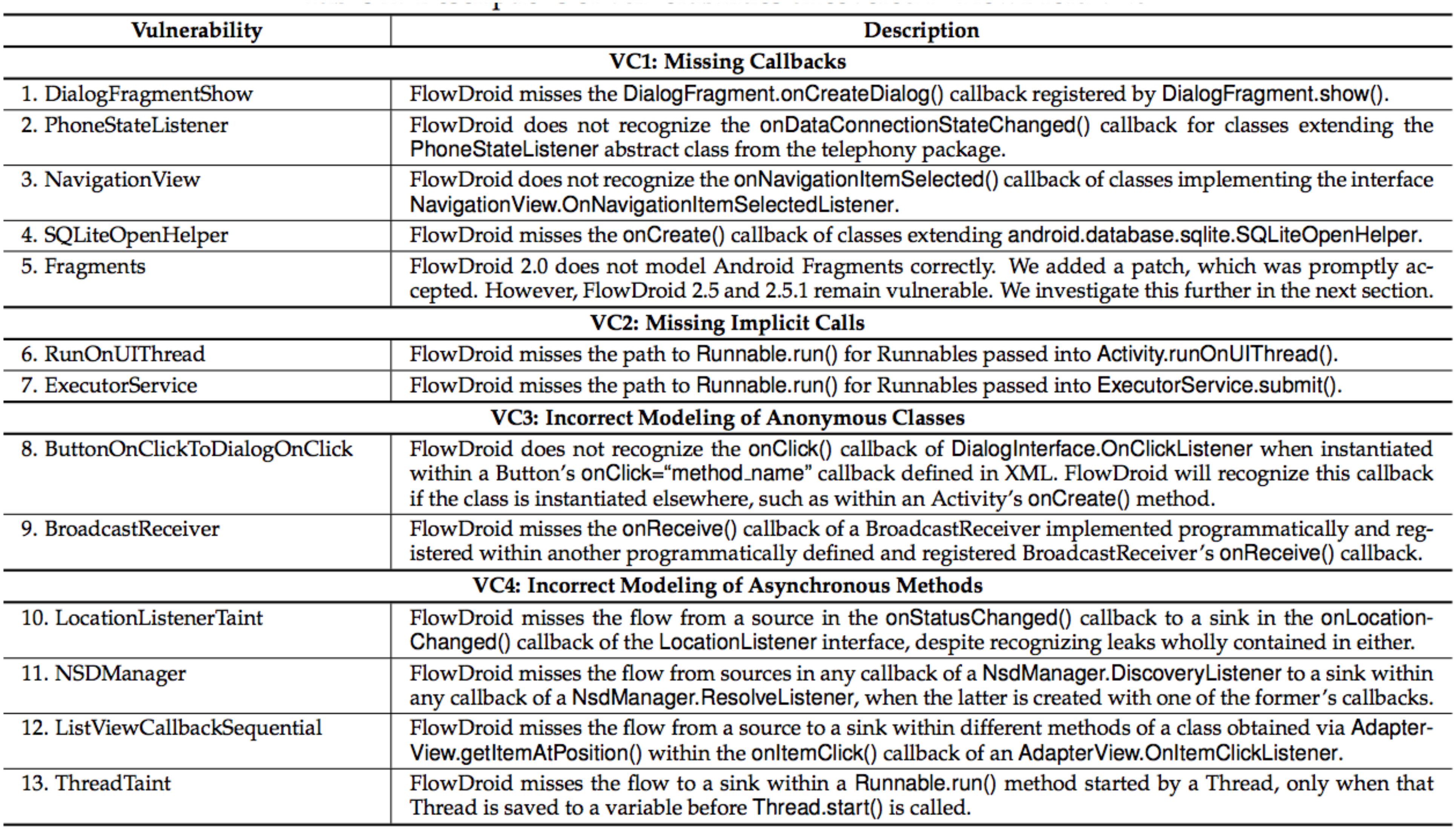


Tool	Undetected Leaks	Undetected Leaks (%)
FlowDroid v2.0	987/2026	48.7%
Argus	1480/2026	73.1%
DroidSafe	83/2026	4.1%

How to get from: *1000s of undetected mutants → unsound choices?*

- Manual Analysis for undetected leaks, using a *systematic approach*
  1. Locate the *source and sink*
  2. Analyze the *call-chain*: Which call (or call sequence) could not be modeled by the analysis?
  3. Build a *minimal working example* with the identified call sequence and test again. On failure to evade detection, go back to 2.

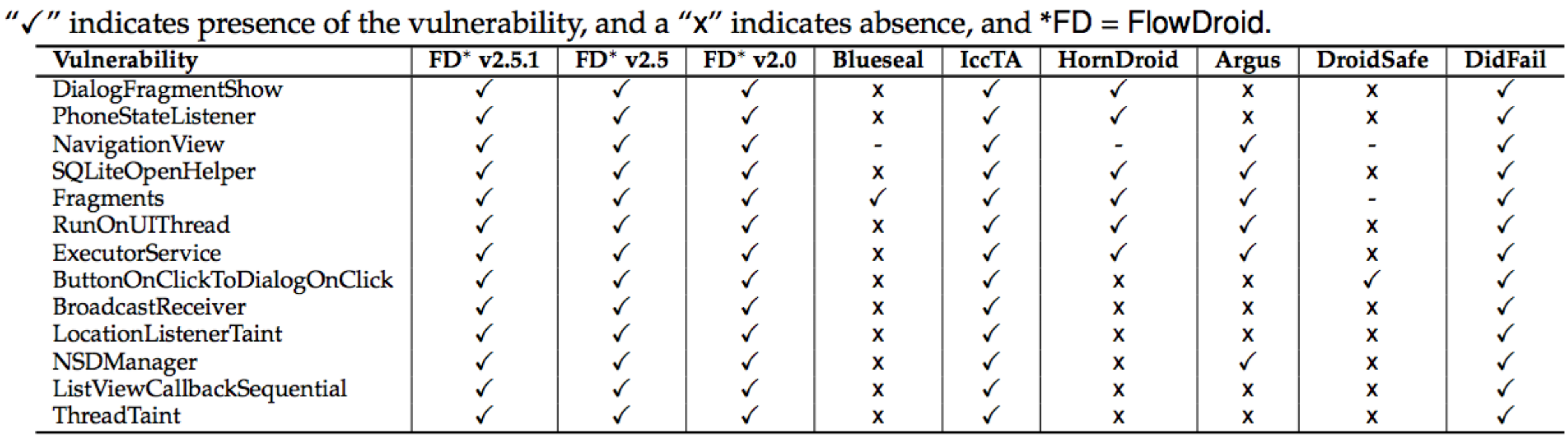
Unsound choices/ flaws



Vulnerability	Description
VC1: Missing Callbacks	
1. DialogFragmentShow	FlowDroid misses the DialogFragment.onCreateDialog() callback registered by DialogFragment.show().
2. PhoneStateListener	FlowDroid does not recognize the onDataConnectionStateChanged() callback for classes extending the PhoneStateListener abstract class from the telephony package.
3. NavigationView	FlowDroid does not recognize the onNavigationItemSelected() callback of classes implementing the interface NavigationView.OnNavigationItemSelectedListener.
4. SQLiteOpenHelper	FlowDroid misses the onCreate() callback of classes extending android.database.sqlite.SQLiteOpenHelper.
5. Fragments	FlowDroid 2.0 does not model Android Fragments correctly. We added a patch, which was promptly accepted. However, FlowDroid 2.5 and 2.5.1 remain vulnerable. We investigate this further in the next section.
VC2: Missing Implicit Calls	
6. RunOnUIThread	FlowDroid misses the path to Runnable.run() for Runnables passed into Activity.runOnUIThread().
7. ExecutorService	FlowDroid misses the path to Runnable.run() for Runnables passed into ExecutorService.submit().
VC3: Incorrect Modeling of Anonymous Classes	
8. ButtonOnClickToDialogOnClick	FlowDroid does not recognize the onClick() callback of DialogInterface.OnClickListener when instantiated within a Button's onClick="method_name" callback defined in XML. FlowDroid will recognize this callback if the class is instantiated elsewhere, such as within an Activity's onCreate() method.
9. BroadcastReceiver	FlowDroid misses the onReceive() callback of a BroadcastReceiver implemented programmatically and registered within another programmatically defined and registered BroadcastReceiver's onReceive() callback.
VC4: Incorrect Modeling of Asynchronous Methods	
10. LocationListenerTaint	FlowDroid misses the flow from a source in the onStatusChanged() callback to a sink in the onLocationChanged() callback of the LocationListener interface, despite recognizing leaks wholly contained in either.
11. NSDManager	FlowDroid misses the flow from sources in any callback of a NsdManager.DiscoveryListener to a sink within any callback of a NsdManager.ResolveListener, when the latter is created with one of the former's callbacks.
12. ListViewCallbackSequential	FlowDroid misses the flow from a source to a sink within different methods of a class obtained via AdapterView.getItemAtPosition() within the onItemClick() callback of an AdapterView.OnItemClickListener.
13. ThreadTaint	FlowDroid misses the flow to a sink within a Runnable.run() method started by a Thread, only when that Thread is saved to a variable before Thread.start() is called.

What about propagation?

- Most flaws propagate (e.g., IccTA and DidFail are completely vulnerable to the same flaws as FlowDroid)
- Some tools only *conceptually* inherit FlowDroid, but use other techniques that preclude same flaws (e.g., DroidSafe, BlueSeal)
  - However, they *may have other flaws*



"✓" indicates presence of the vulnerability, and a "x" indicates absence, and *FD = FlowDroid.

Vulnerability	FD* v2.5.1	FD* v2.5	FD* v2.0	Blueseal	IccTA	HornDroid	Argus	DroidSafe	DidFail
DialogFragmentShow	✓	✓	✓	x	✓	✓	x	x	✓
PhoneStateListener	✓	✓	✓	x	✓	✓	x	x	✓
NavigationView	✓	✓	✓	-	✓	-	✓	-	✓
SQLiteOpenHelper	✓	✓	✓	x	✓	✓	✓	x	✓
Fragments	✓	✓	✓	✓	✓	✓	✓	-	✓
RunOnUIThread	✓	✓	✓	x	✓	✓	✓	x	✓
ExecutorService	✓	✓	✓	x	✓	✓	✓	x	✓
ButtonOnClickToDialogOnClick	✓	✓	✓	x	✓	x	x	✓	✓
BroadcastReceiver	✓	✓	✓	x	✓	x	x	x	✓
LocationListenerTaint	✓	✓	✓	x	✓	x	x	x	✓
NSDManager	✓	✓	✓	x	✓	x	✓	x	✓
ListViewCallbackSequential	✓	✓	✓	x	✓	x	x	x	✓
ThreadTaint	✓	✓	✓	x	✓	x	x	x	✓

Parts of a paper

- Parts of paper (vast generalization)

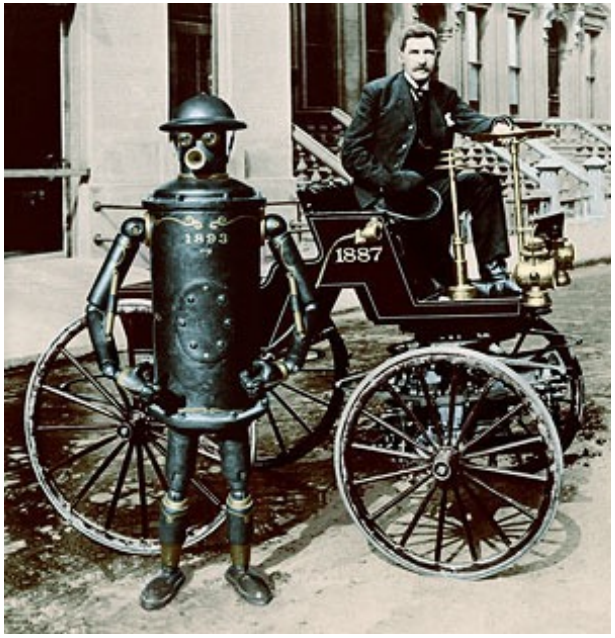
1. Abstract
2. Introduction
3. Related Work/Background
4. Solution/Problem
5. Evaluation/Analysis/Experiment
6. Discussion (often, but not always)
7. Conclusions

Abstract

- One sentence each for:
  - Area
    - Topic of work
  - Problem
    - What's the issue?
  - Solution
    - How do you propose to address the problem?
  - Methodology
    - What's the experiment?
  - Results
    - What did you find?
  - Take Away: Lesson

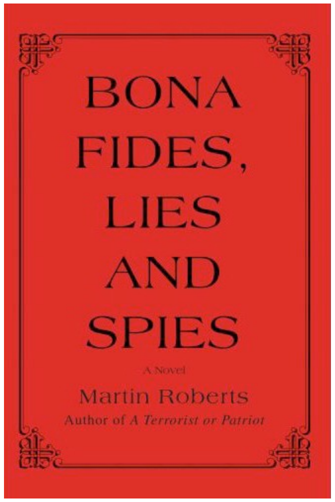
Introduction

- One paragraph each on:
- Area
  - More elaborate
- Problem
  - Scenario
- Why is problem not solved
  - Brief of related work or the challenge
- Proposed insight (“In this paper, ...”)
  - What is the experiment?
- Contributions -- What will the reader learn?
- Boilerplate outline (?)



Related work

- This is a statement of the work that led to this one.
  - who this work relies on
  - who has done work in the area
  - areas that inspired this work (not just technology)
  - Not a laundry list
- There are several reasons for related work section:
  - Motivate the current work
  - Differentiate from past work
  - Establish "bona fides"



BONA FIDES, LIES AND SPIES

A Novel

Martin Roberts

Author of *A Terrorist or Patriot*

Motivation, Background

- **Motivation**
  - Why is this a problem?
  - Motivating Example: Alice…
  - Why isn't the problem solved?
    - Forward/backward reference to the related work.
- **Problem, assumptions**: Problem statement, threat model, TCB.
- **Background:** What all does the reader need to know to understand your approach?
    - Already known material related to the solution
    - Tip: You can always move text from the design to the background, to focus on the *novel contributions in the design*.

System Architecture and Design

- How do you solve the problem?
- General Architecture / Overview
- What are the
  - Design Goals?
  - Challenges?
  - Contributions of your design (i.e., the design decisions) that help overcome the design challenges, hence achieving the design goals?

Experiment

- Experiment
    - Means of showing truth
    - Big Insight -- Hypothesis -- Claim
        - Show why it is interesting
    - Expected Results
        - Informal proof/argument that is true
- Experiment types
    - *Empirical* - measure some aspect of the solution
    - *Analytical* - prove something about solution
    - *Observational* - show something about solution

Results vs Findings

- Results
    - Summarize -- what do the results mean?
    - Specific experiments
        - We did X, saw Y
    - What do the experiments prove
    - What other experiments would you want to do based on these results?
- Key Findings
    - What do the results mean?
    - What are the lessons?
    - Lead to the takeaway.

Conclusion

- Like the abstract in past tense
- Problem
  - What was the problem?
- Solution
  - What was the insight and why was it expected to work?
- Method and Results
  - What did you find?
- Take away: Lesson
- Future work

Hint

- Intro: tell them what you are going to tell them
- Body: tell them
- Conclusion: tell them what you told them.

The End