From Dalvik Bytecode Analysis to Leak Detection in Android Applications

Alexandre Bartel, Eric Bodden, Steven Artz, Siegfried Rasthofer

1st ICFEM Workshop on Default Privacy
Thursday 6 November 2014
Luxembourg, Luxembourg

Evolution of Phones

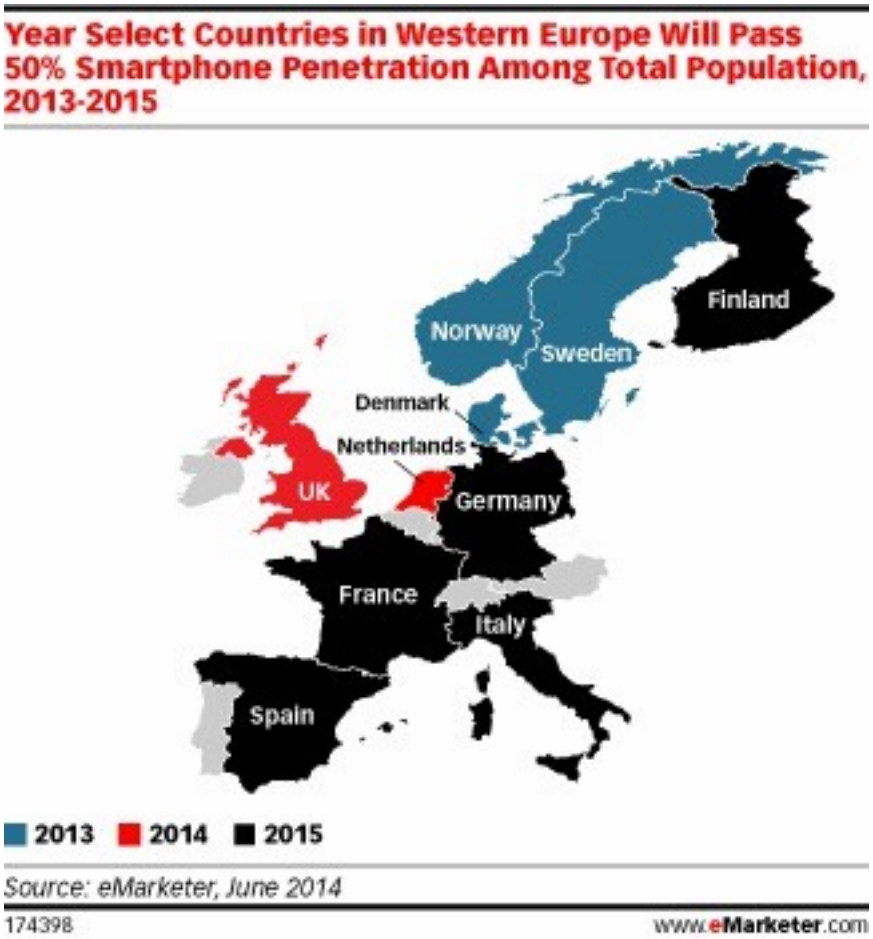
1985	1995	2005	2015
10,000	100,000	1,000,000	10,000,000 loc

“Smart”Phone = Computer + Sensors + Apps

CASED

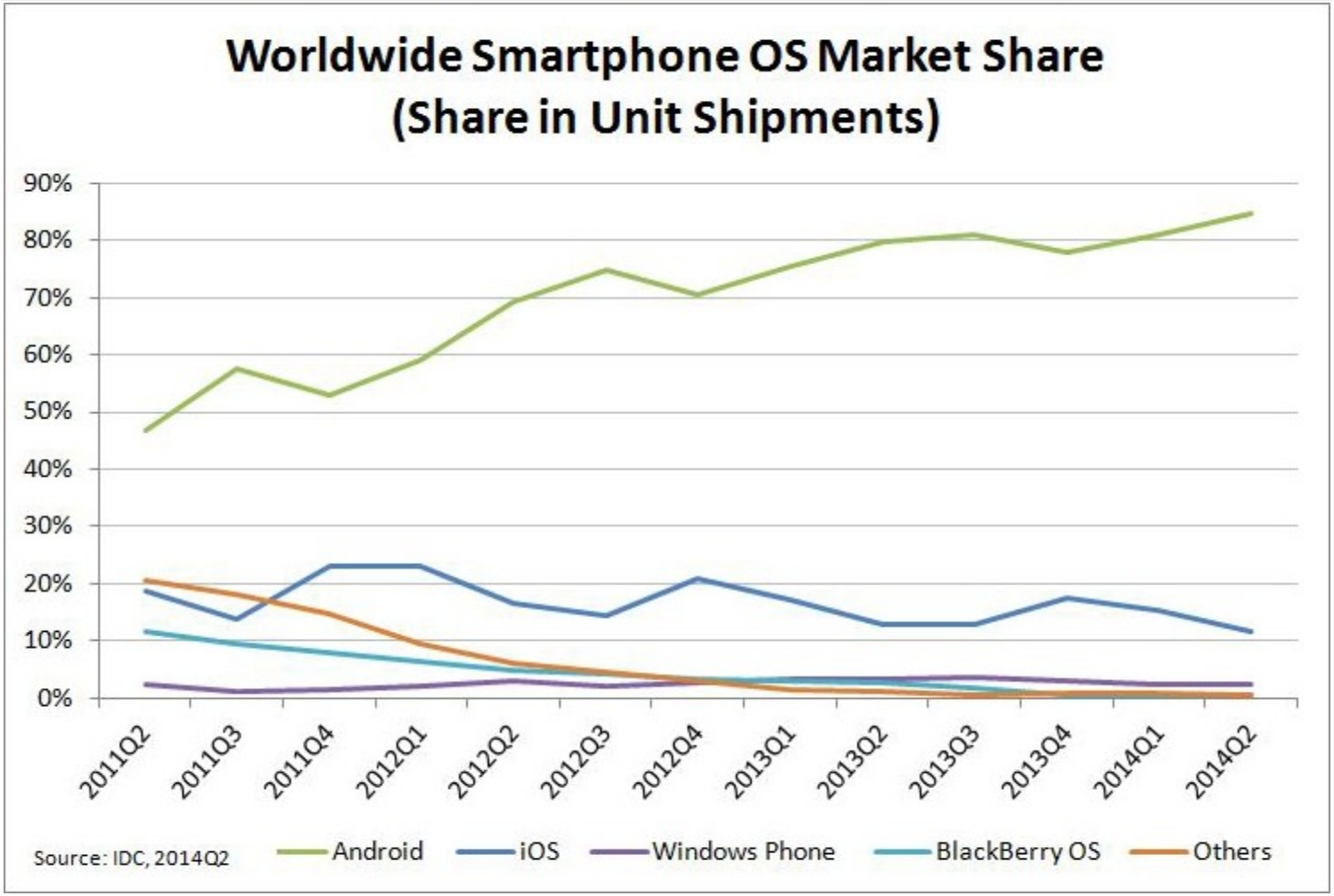
TECHNISCHE UNIVERSITÄT DARMSTADT
Fraunhofer SIT
h da HOCHSCHULE DARMSTADT UNIVERSITY OF APPLIED SCIENCES
LOEWE – Landes-Offensive zur Entwicklung Wissenschaftlich-ökonomischer Exzellenz

Smartphone Penetration



Personal Information Stored on Smartphones

Android Market Share > 80%!



Popular Android Apps Leaking S
Report Finds

By Chloe Albanesius | October 22,

Skype for Android leaks sensitive data

WhatsApp leaks use
and messages

19 MAY 2011 APPLICATIONS

eaking

velopers may

Angry Birds and other Mobile Gaming apps leaking your private information to NSA

by **Swati Khandelwal** on Monday, January 27, 2014

Hackers

By DANIEL BATE

PUBLISHED: 10:13

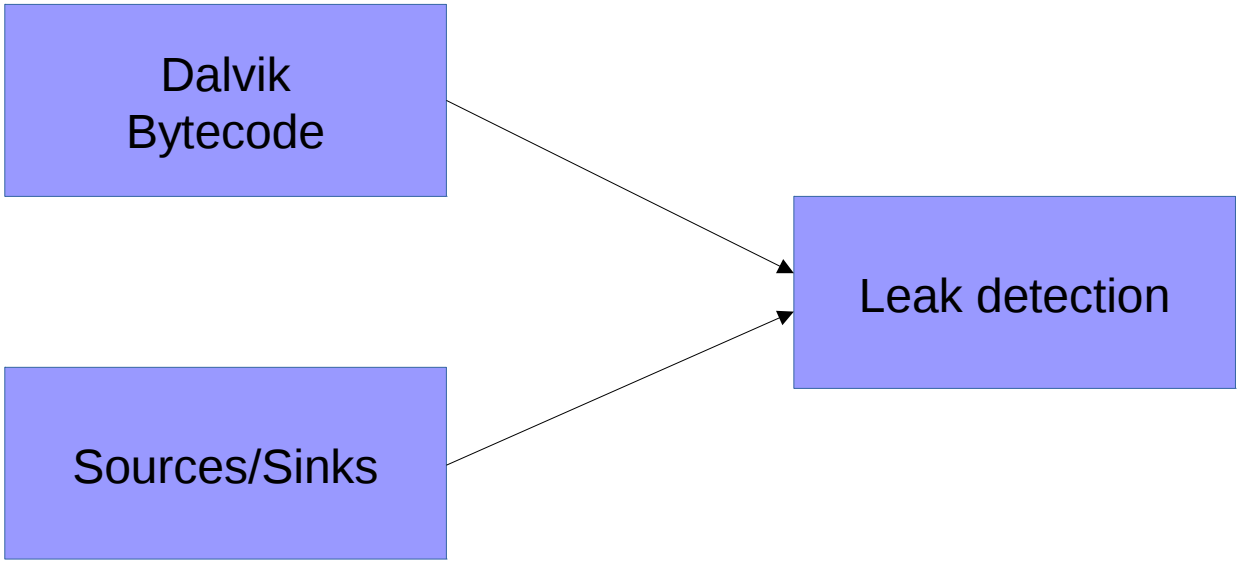
Published January

Appthority Security Team

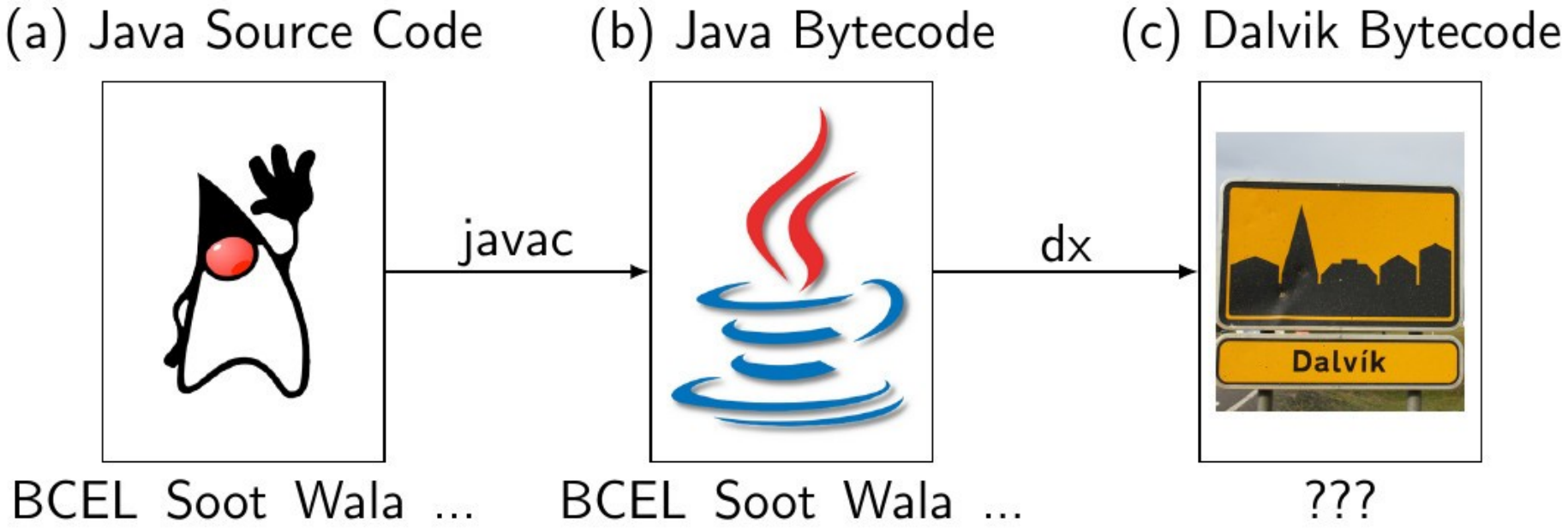
d apps leak user privacy data

find permitted apps transmit phone numbers, location, and SIM card IDs

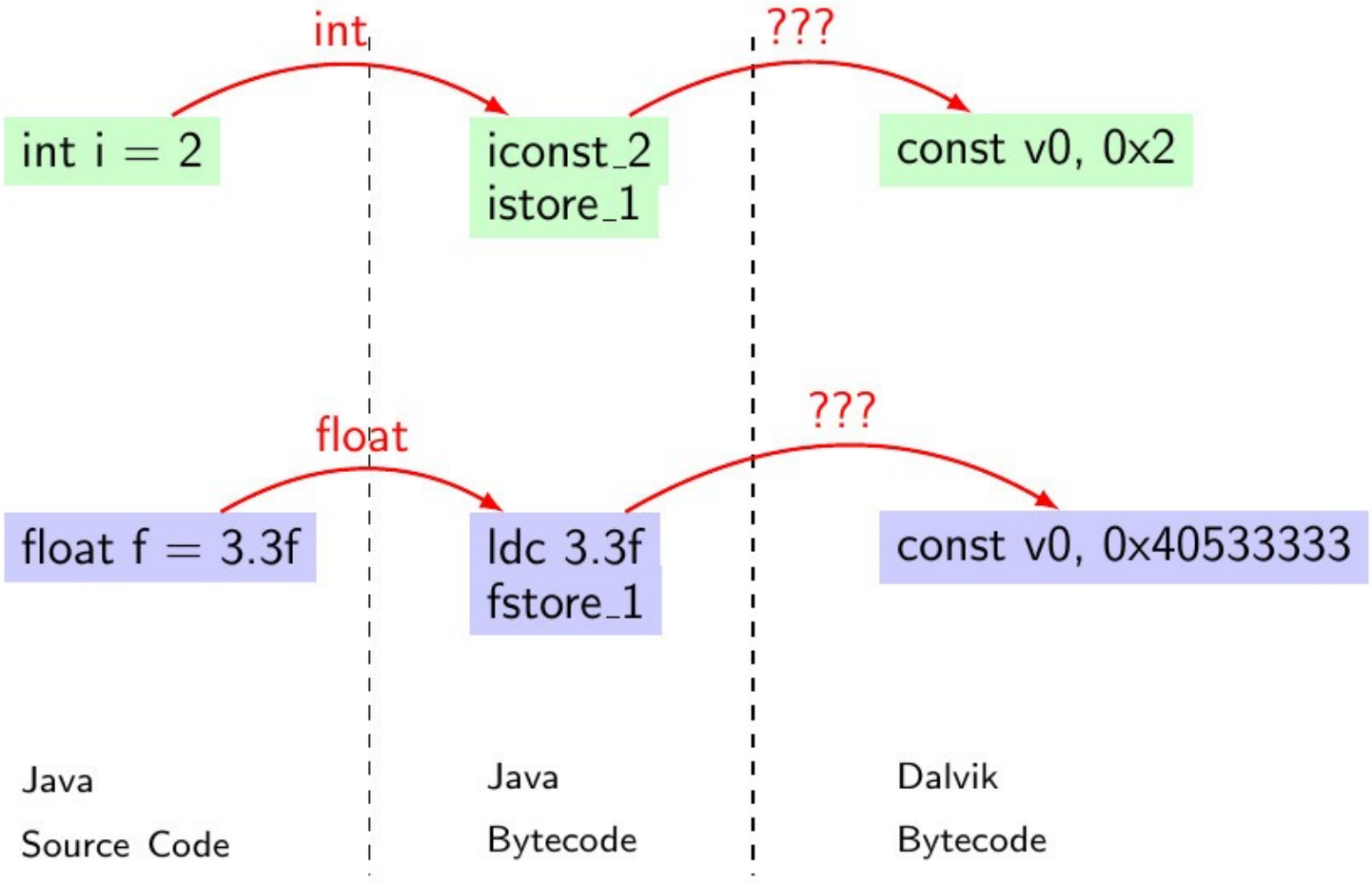
Overview



How to Analyze Dalvik Bytecode?



Problem: Type Information is Missing



Solution: Find the Missing Information!

CASED

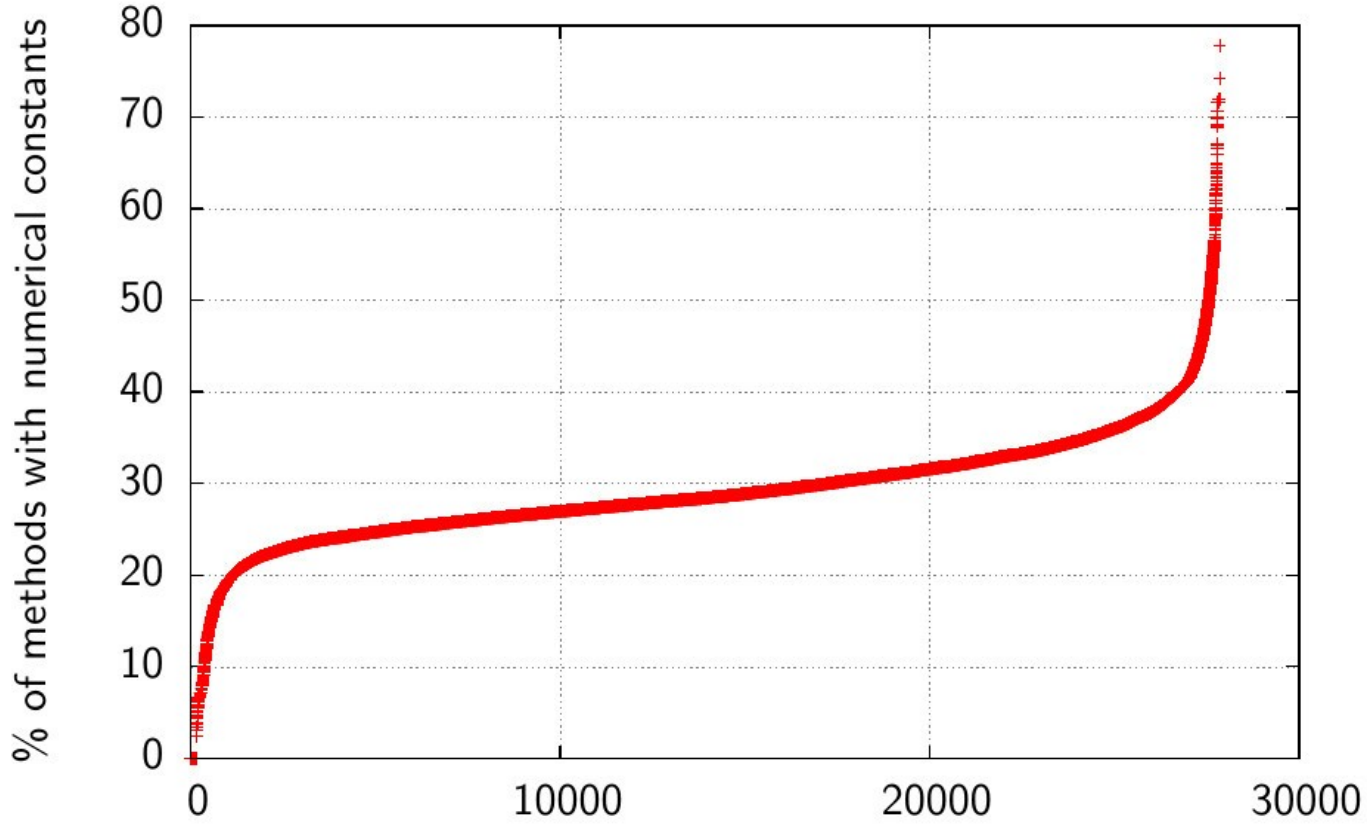
```
public float untyped(float[] array, boolean flag) {
    if (flag) {
        float delta = 0.5f
        return array[7] + delta
    } else {
        return 0.2f
    }
}
```

```
UntypedSample.untyped:([FZ)F
        0000: if-eqz v4, 0009
        0002: const/high16 v0, #0x3f000000
        0004: const/4 v1, #0x7
        0005: aget v1, v3, v1
        0007: add-float/2addr v0, v1
        0008: return v0
        0009: const v5, #0x3e4ccccd
        000c: return v5
```

type is Float!

type is Float!

99.4% of the Apps have Numerical Constants



Evaluation: Do we Correctly Type the Code?

- Set of 27,846 Android applications
- Total of 135,289,314 methods

Our algorithm correctly types

99%

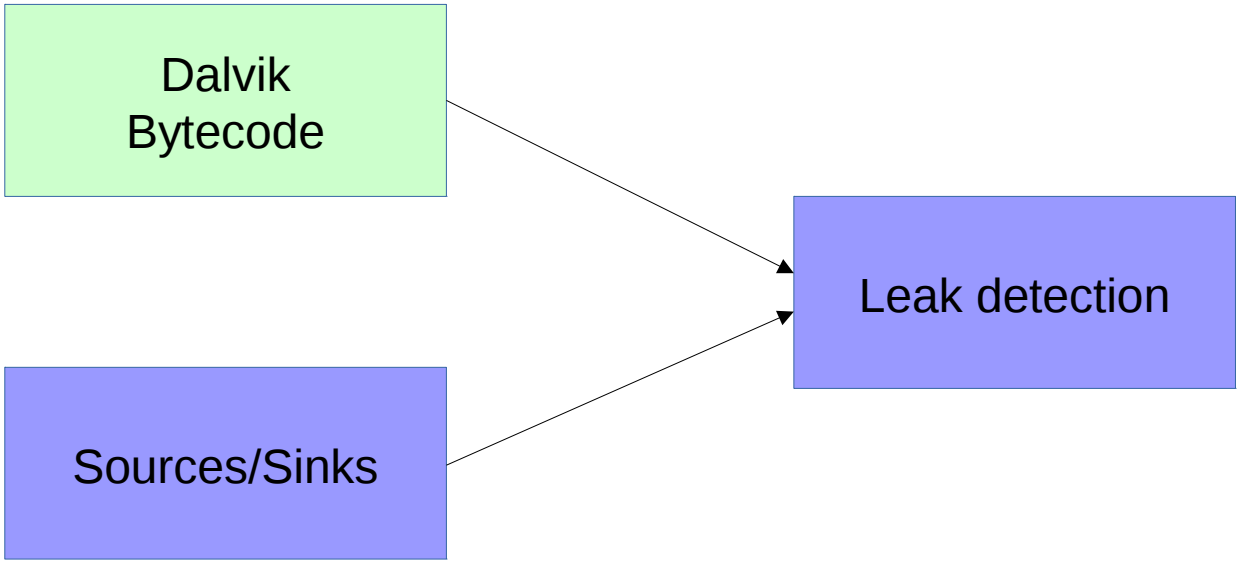
of the analyzed methods

Future Work

- Unresolved reference
- Jump to code in array
- Multiple types for a single variable

Bartel, A., Klein, J., Le Traon, Y., & Monperrus, M. (2012, June). Dexpler: converting android dalvik bytecode to jimple for static analysis with soot. In Proceedings of the ACM SIGPLAN International Workshop on State of the Art in Java Program analysis (pp. 27-38). ACM.

Overview

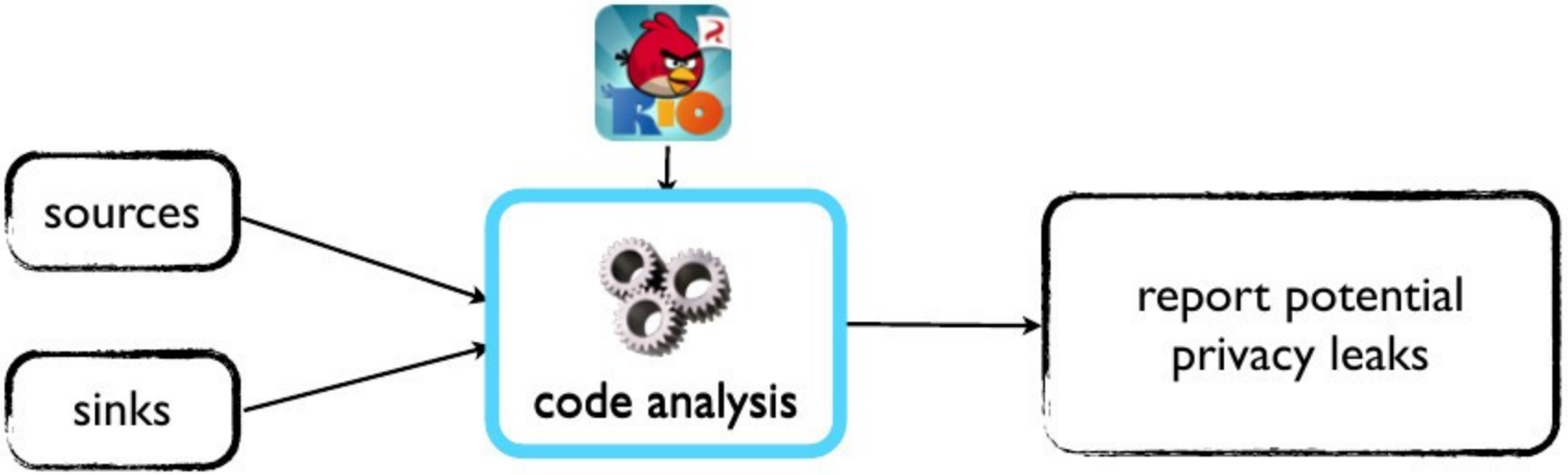


Detecting Privacy Leaks

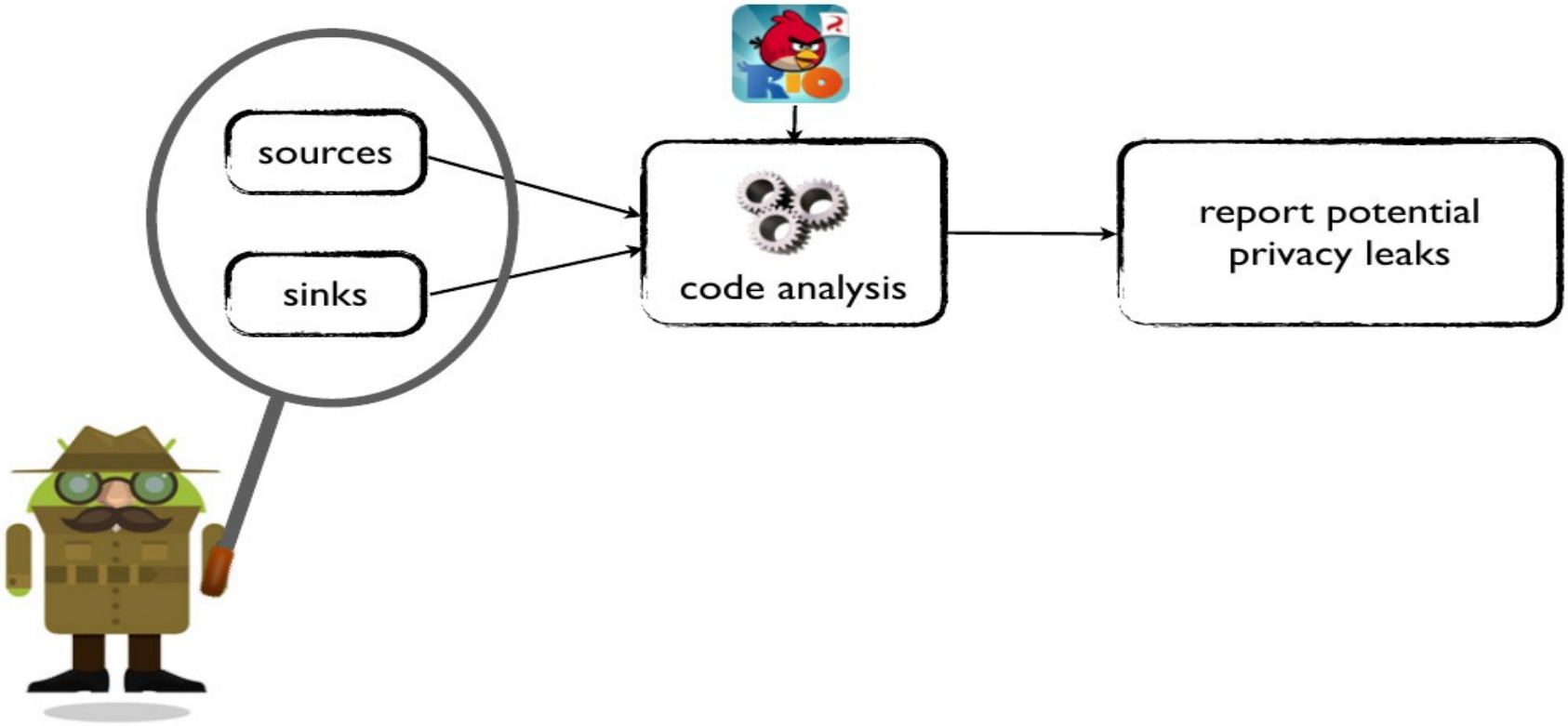
CASED

- Dynamic Approaches:
  - TaintDroid [OSDI'10],
  - Aurasium [USENIX'12],
  - "Dr. Android and Mr. Hide"[SPSM'12],
  - etc.
- Static Approaches:
  - ScanDroid [TR 09],
  - DeD [SEC'11],
  - CHEX [CCS'12],
  - LeakMiner [WCSE'12],
  - ScanDal [Most'12],
  - AndroidLeaks [TRUST'12],
  - SAAF [SAC'13],
  - FlowDroid [PLDI'14],
  - etc.

Detecting Privacy Leaks: Generic Approach



But...

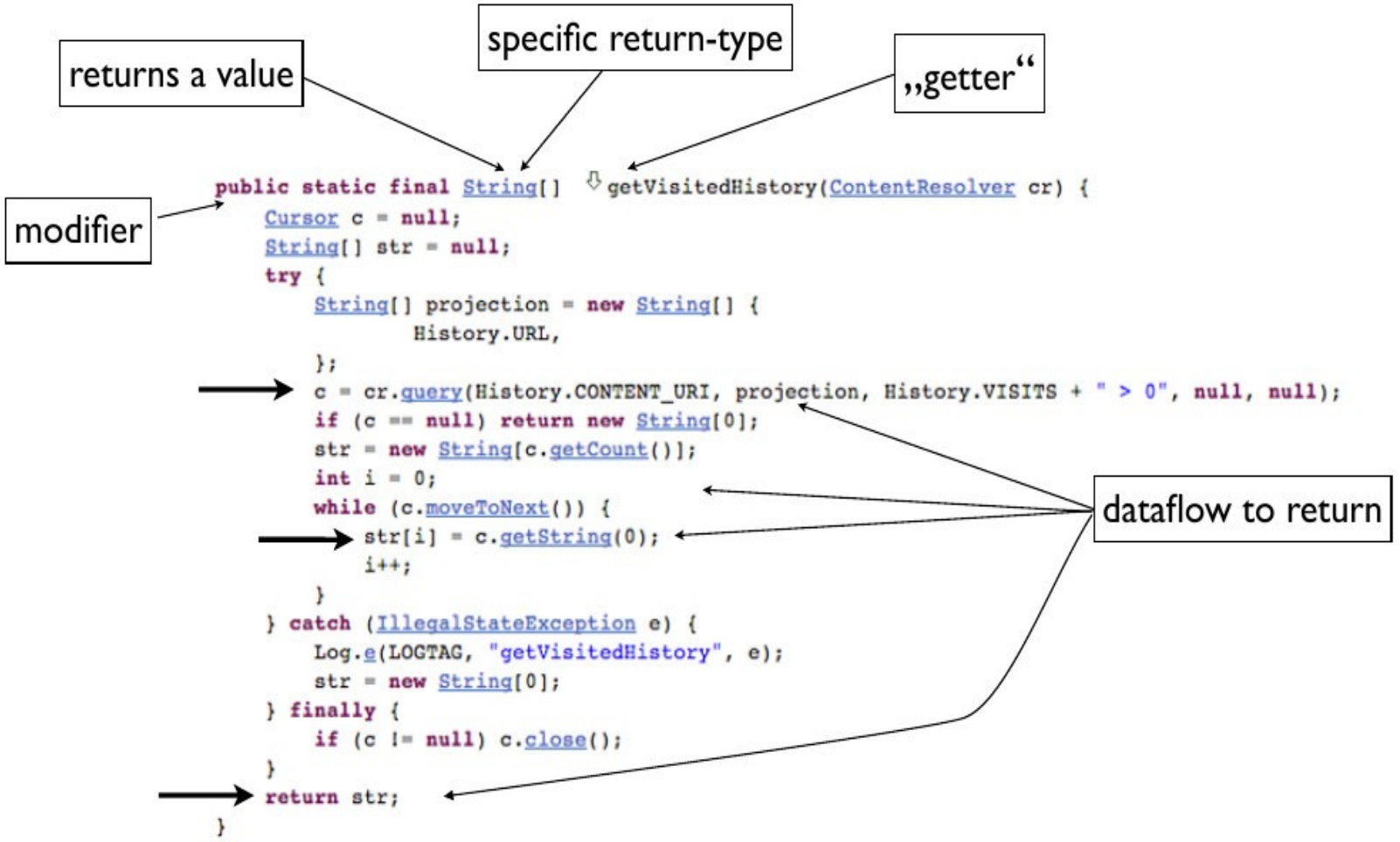

sources
sinks
code analysis
report potential
privacy leaks

Complete List Available?

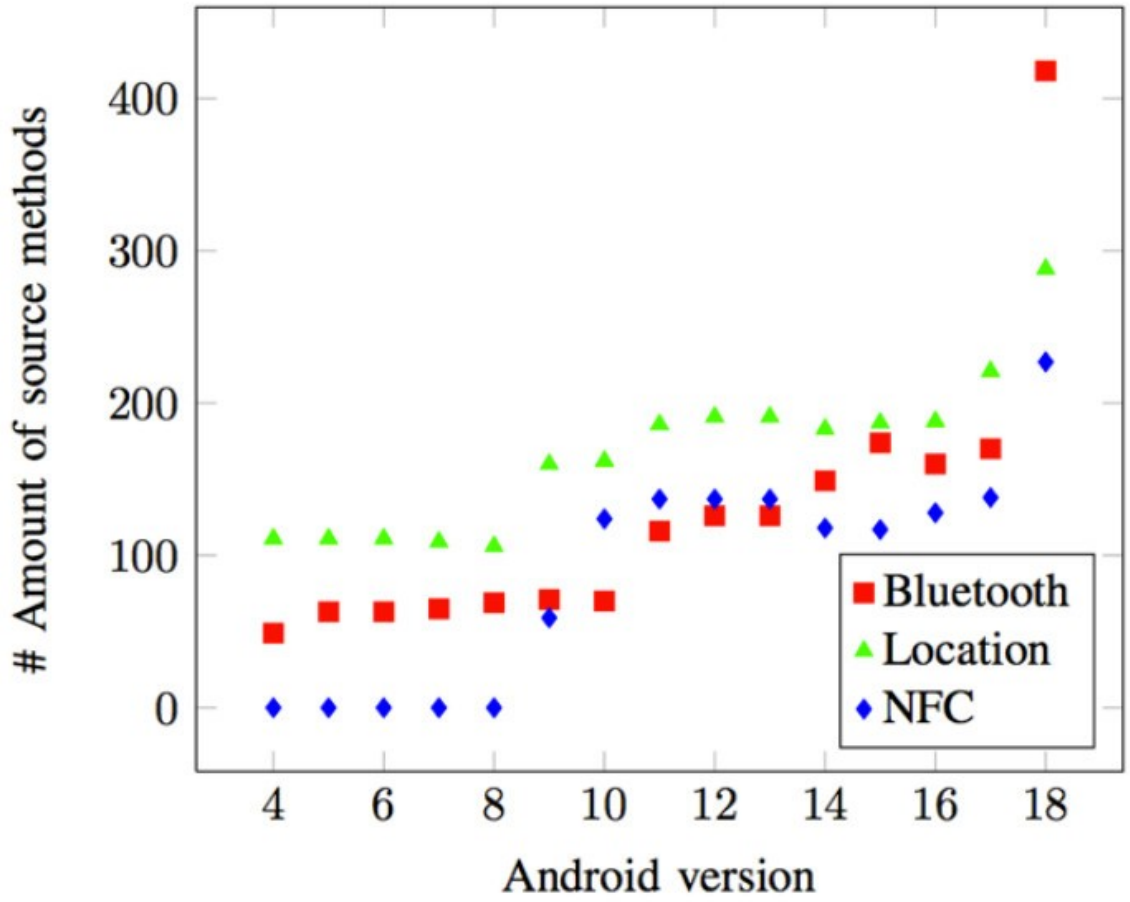
CASED

Method	TaintDroid	SCanDroid	DeD
	?		
Location.getLongitude()	✓	✓	✓
Location.getLatitude()	✓	✓	✓
Browser.getAllBookmarks()	✓		
SmsManager.sendTextMessage	✓	✓	✓
Log.d()			✓
URL.openConnection()	✓		

Machine Learning with Code Features



Evaluation on Android Versions



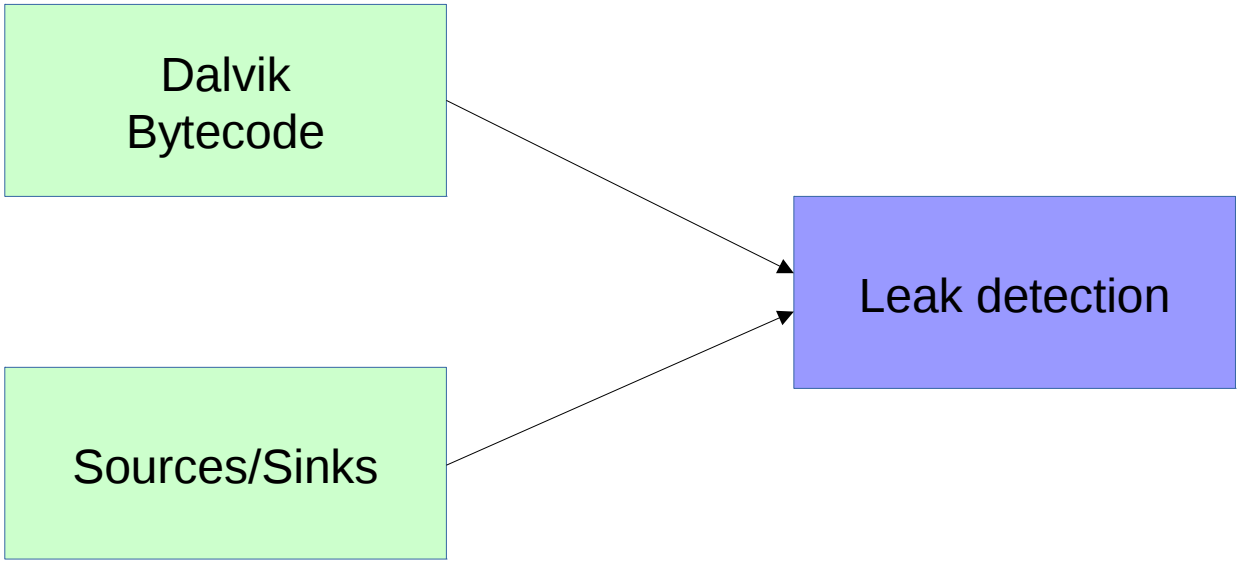
Top Source/Sink Methods in Malware

CASED

Method	TaintDroid	SCanDroid	DeD
BluetoothAdapter.getAddress()	✗	✗	✗
WifiInfo.getMacAddress()	✗	✗	✗
Locale.getCountry()	✗	✗	✗
WifiInfo.getSSID()	✗	✗	✗
GsmCellLocation.getCid()	✗	✗	✗
GsmCellLocation.getLac()	✗	✗	✗
Location.getLongitude()	✓	✓	✓
Location.getLatitude()	✓	✓	✓
Browser.getAllBookmarks()	✓	✗	✗
SmsManager.sendTextMessage	✓	✓	✓
Log.d()	✗	✗	✓
URL.openConnection()	✓	✗	✗

Rasthofer, Siegfried, Steven Arzt, and Eric Bodden. "A machine-learning approach for classifying and categorizing android sources and sinks." 2014 Network and Distributed System Security Symposium (NDSS). 2014.

Overview



FlowDroid

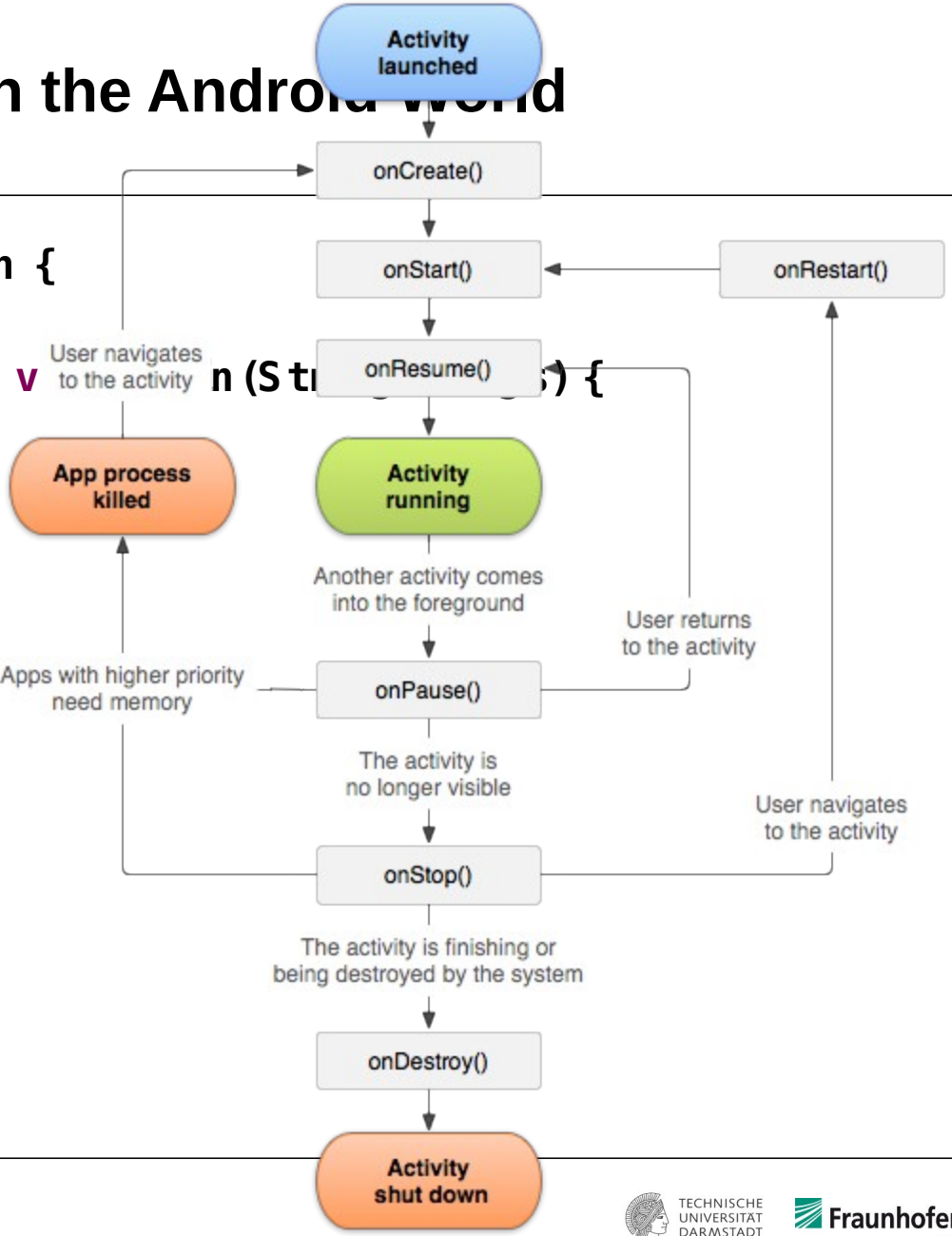
- Challenges in the Android World
- Highly Precise Taint Analysis
  - The Principles
  - Aliasing for Highly Precise Analyses
- Experiments
  - The DroidBench Micro Benchmark Suite

Challenges in the Android World

```
public class Main {

    public static v n(Str ) {
        …
    }

}
```



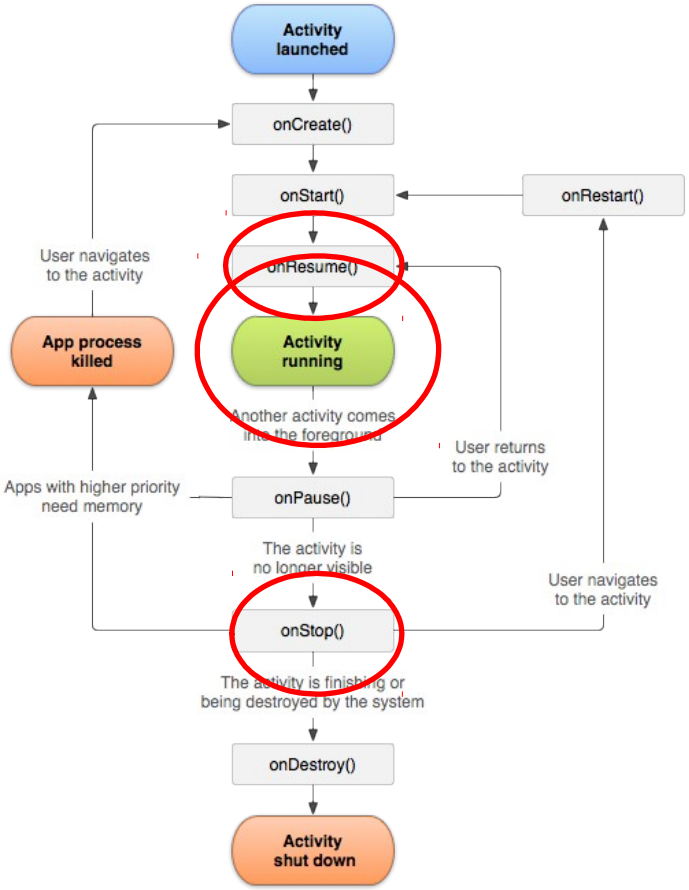
Modeling The Android Lifecycle

- Model Lifecycle Through Dummy Main Method
- Use Opaque Predicates and Jumps
  - All paths allowed in spec must be possible in method
  - Lots of paths, but doesn't matter (see later)

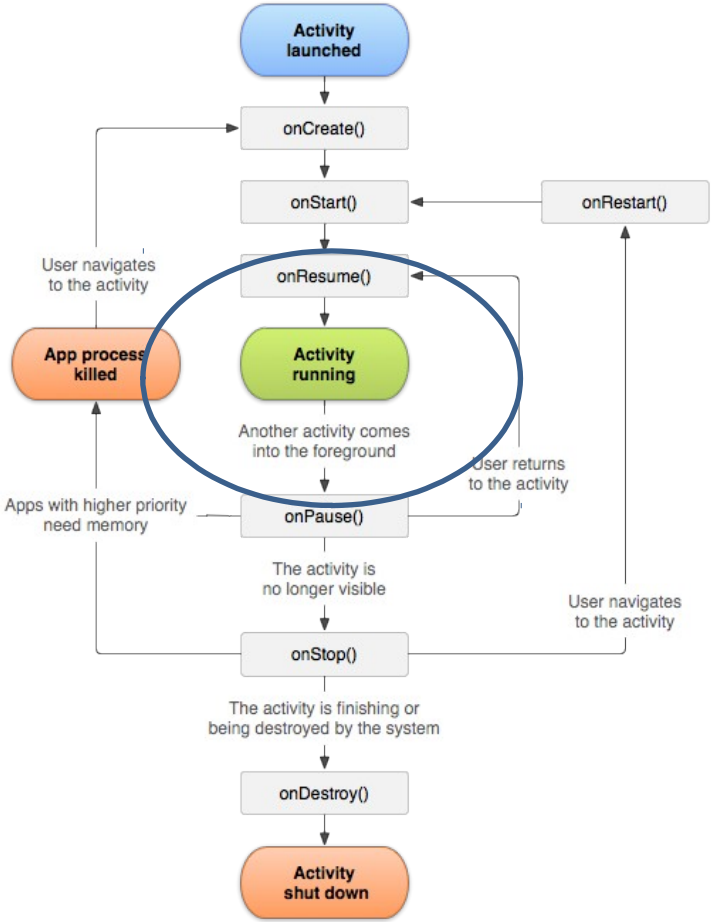
Modeling The Android Lifecycle

```
i= 0;
l1:if(i== 0)goto l9; //Skip the activity

Activity1 act1 = new com.ext.Activity1();
act1.onCreate(…);act1.onStart;
l2:act.onResume();
…
act1.onPause(…);
l1:if(i== 1)goto l2;
act1.onStop();
act1.onDestroy();
if(i== 2)goto l1; //Run activity again
```



Challenges in the Android World



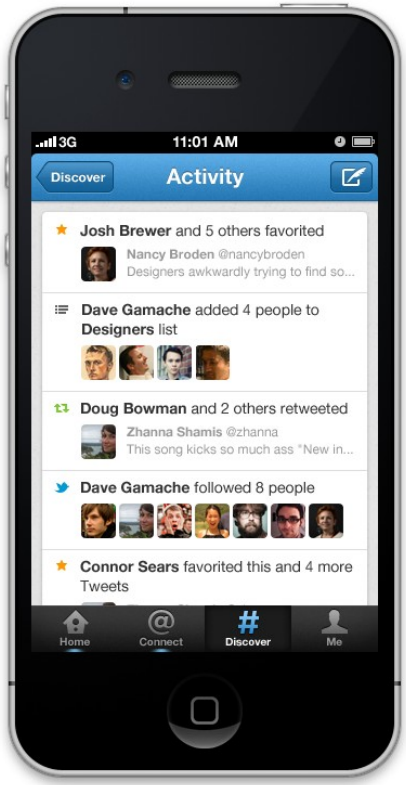
Challenges in the Android World

CASED

onLocationChanged

onGpsStatusChanged

onSensorChanged



onLowMemory

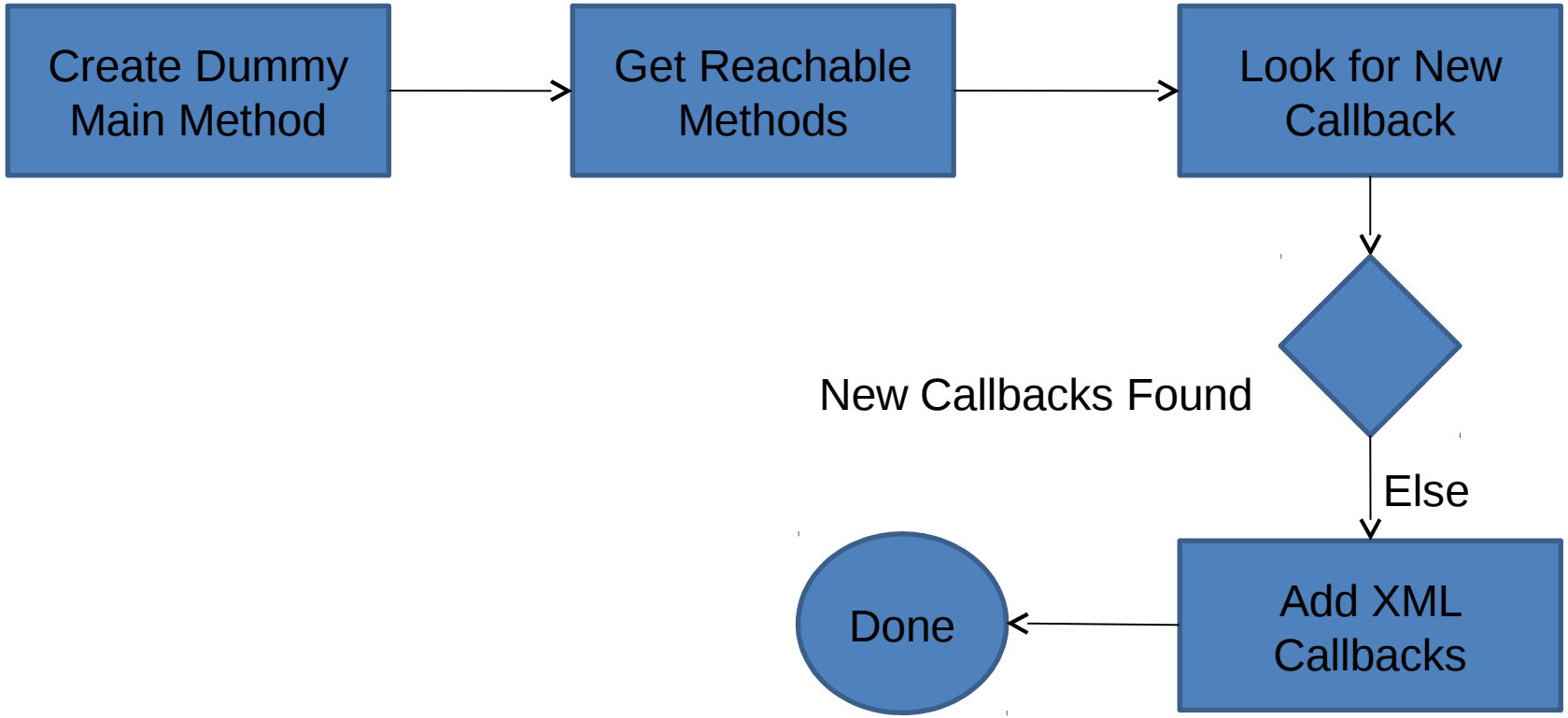
onGesture

onZoomChange

Modeling Callbacks

- Same Technique as for Lifecycle
  - Call callback methods in dummy main method
- Simplification: Callbacks never die
  - Registered from app start till termination
- Not as Easy as it Sounds
  - Callbacks that register new callbacks
  - Callbacks defined in XML files

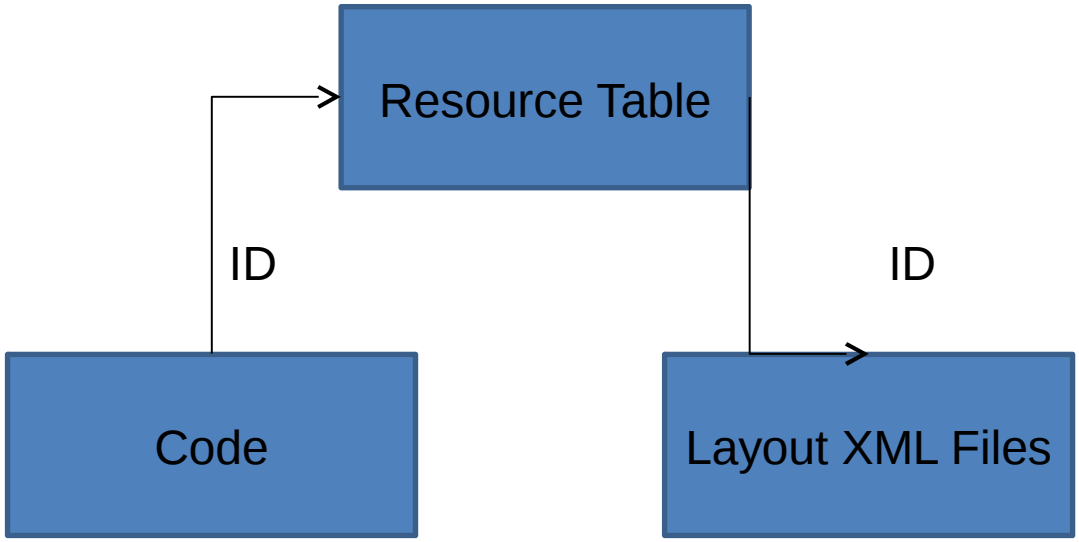
Modeling Callbacks



Challenges in the Android World

- Many Sources and Sinks
  - API methods from the SuSi list (NDSS'14)
  - User Interface Controls (e.g., Password Fields)
- Scalability Issue When Running With All Sources/Sinks
  - Piggybacking source on taint abstraction won't scale
- The Android Framework is Huge
  - Analyzing the framework with every app doesn't scale
  - Need library abstractions

Dissecting Android Apps: Layout Files



Dissecting Android Apps: Layout Files

CASED

1. Parse The Global Resource Table

2. Parse The Layout XML Files

3. For every Layout File:
    1. Scan the code for registrations of the component ID
    2. Lookup the method ID to get the name
    3. Add the handler to the dummy main method

Highly Precise Taint Tracking

CASED

- Based on the IFDS Framework by Reps and Horwitz
  - Idea: Data flow programs reduced to graph reachability

- Field-Sensitive
- Object-Sensitive
- Flow-Sensitive
- Context-Sensitive
  - Unlimited Depth!
  - Fix-Point iteration until no new callee-s

And what about aliasing?

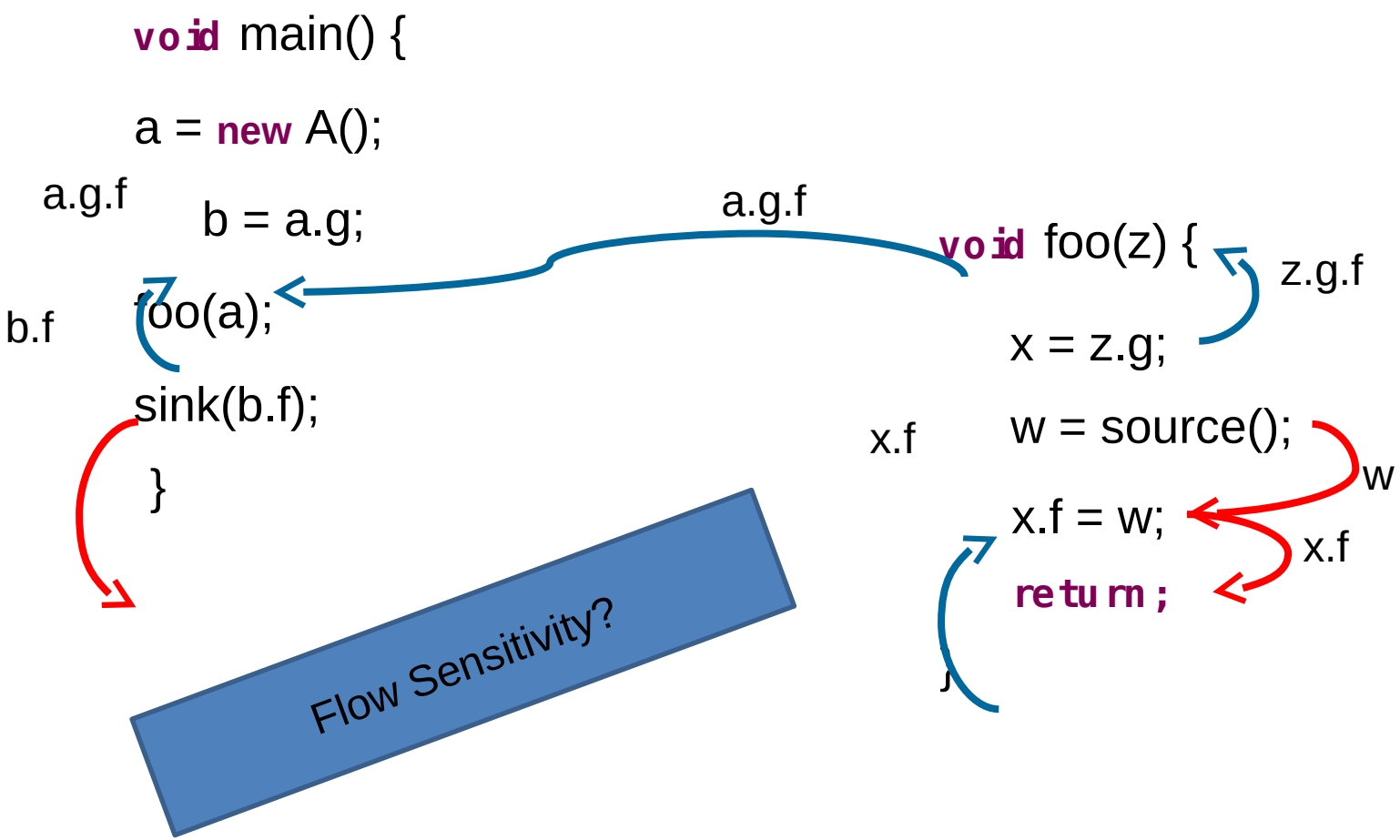
Highly Precise Taint Tracking

CASED

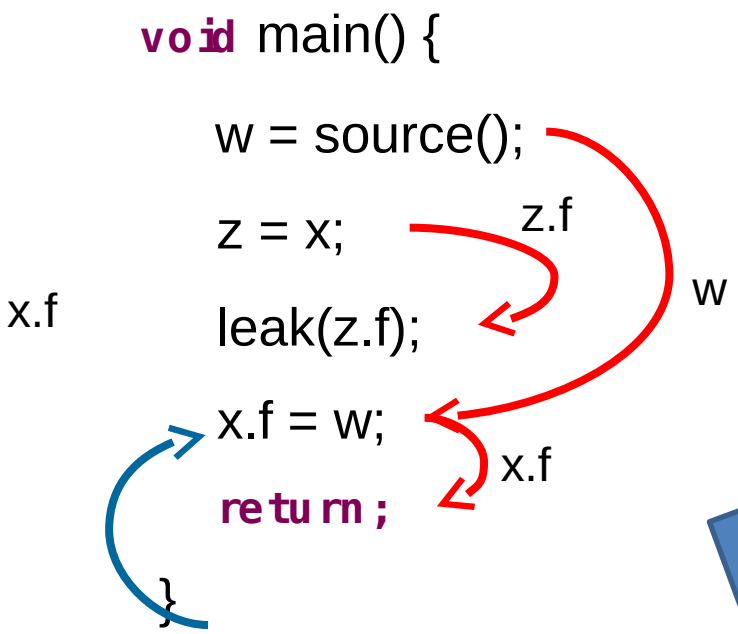
- Need an Alias Analysis With Same Precision
- Upfront Analysis Does Not Scale
- Solution: On-Demand Alias Analysis
  - Idea: Re-use same IFDS-based analysis
  - Two interleaved solvers
  - Technique adapted from Andromeda by Tripp et al.

(in: Fundamental Approaches to Software Engineering)

Highly Precise Taint Tracking



Highly Precise Taint Tracking



Need to remember when taint becomes "live"

We call it "Activation Statement"

DroidBench – Benchmarks for Android

CASED

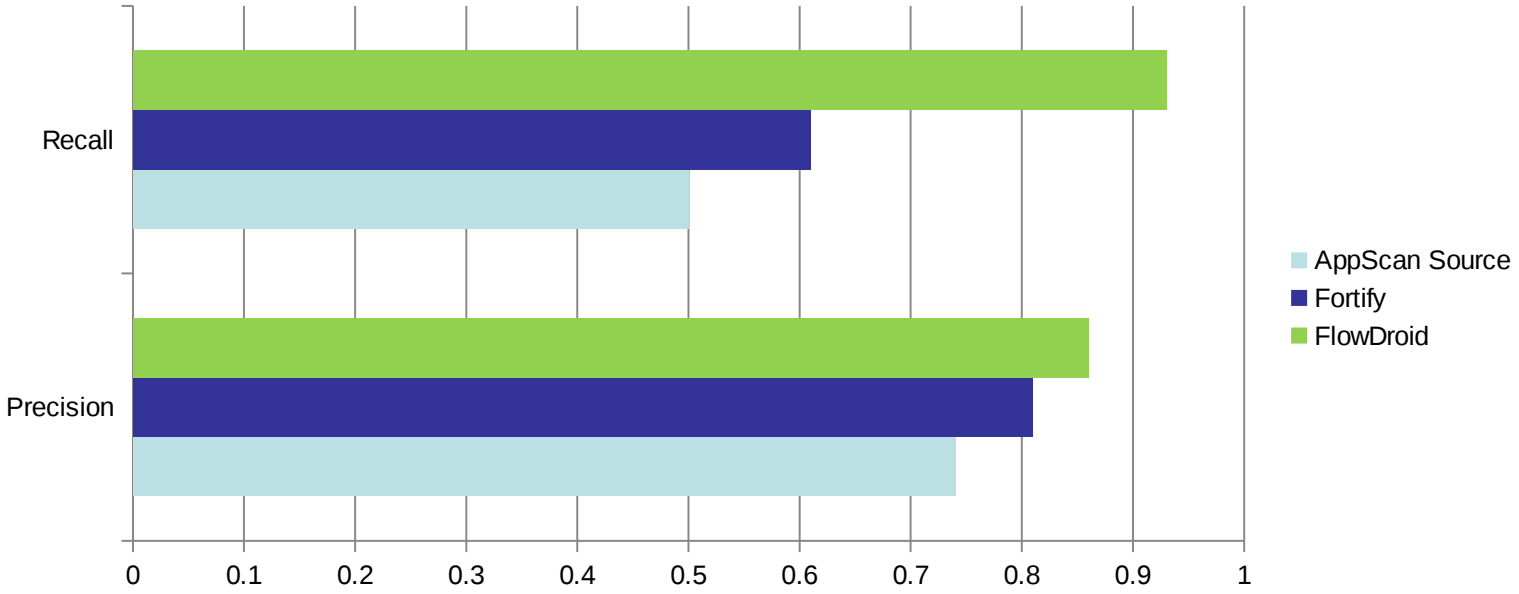
- Compare Static/Dynamic Analysis Tools for Android
- Open Source
- You’re Welcome to Contribute!

DroidBench – Benchmarks for Android

CASED

- 64 Test Cases So Far
  - Arrays and Lists
  - Callbacks
  - Field And Object Sensitivity
  - Inter-App Communcation
  - Lifecycle
  - General Java
  - Miscellaneous Android-Specific
  - Implicit Flows
  - Reflection

FlowDroid vs. The Rest on DroidBench



Future Work

- Native Code
  - Currently under-approximated by default
  - NativeCallHandler interface for custom implementations

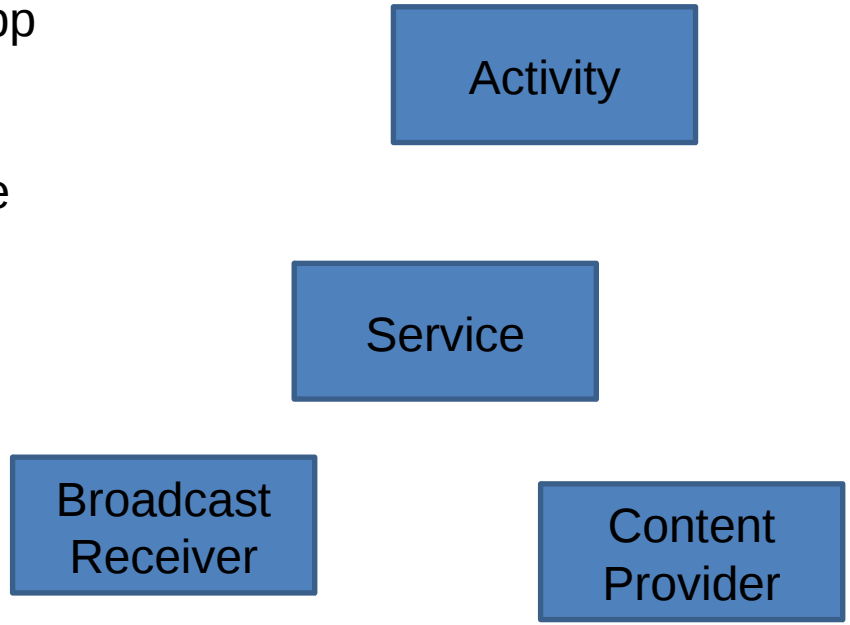
- Library Functions
  - TaintPropagationHandler interface
  - Default implementation: Simple rules
  - More clever solution under submission

- More Efficient Callgraph Algorithms

Future Work

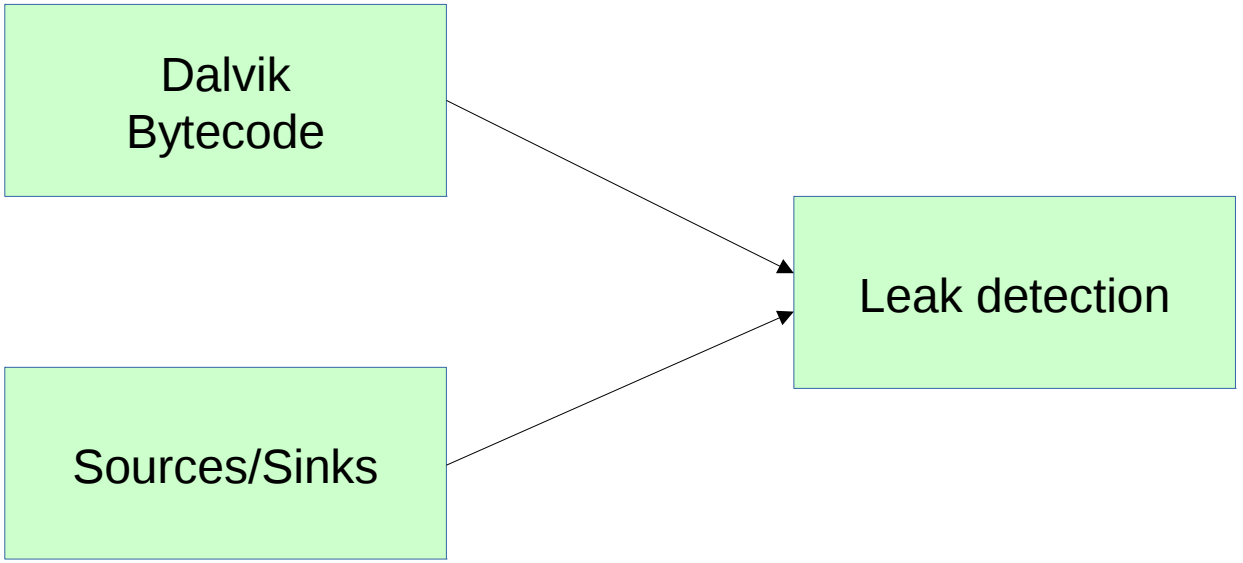
- Inter-Component Communication
  - 320 different activities in Facebook app
  - Support for static fields

  - Communication using intents possible
  - Solution under submission



Arzt, S., Rasthofer, S., Fritz, C., Bodden, E., Bartel, A., Klein, J., ... & McDaniel, P. (2014, June). Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps. In Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation (p. 29). ACM.

Overview



The End

Dexpler	http://www.abartel.net/dexpler/
Soot	https://github.com/Sable/soot
SuSi	http://sseblog.ec-spride.de/tools/susi/
FlowDroid	http://sseblog.ec-spride.de/tools/flowdroid/
Epicc	http://siis.cse.psu.edu/epicc/
IccTA	https://sites.google.com/site/icctawebpage/
DroidForce	https://github.com/secure-software-engineering/DroidForce

Alexandre Bartel
Center for Advanced Security Research Darmstadt (CASED)
Secure Software Engineering Group (EC-SPRIDE)
Email: alexandre.bartel@cased.de