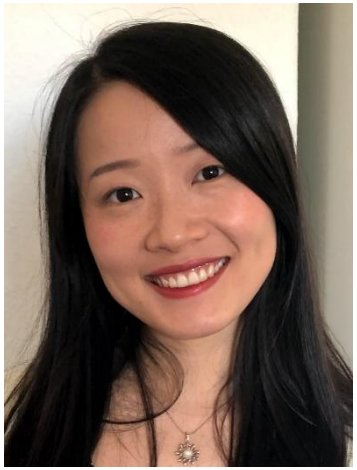
HEINZ NIXDORF INSTITUT
UNIVERSITÄT PADERBORN

A Qualitative Analysis of Android Taint-Analysis Results

ASE 2019, San Diego

Linghui Luo

@LinghuiLuo



Linghui Luo

PADERBORN UNIVERSITY

Eric Bodden

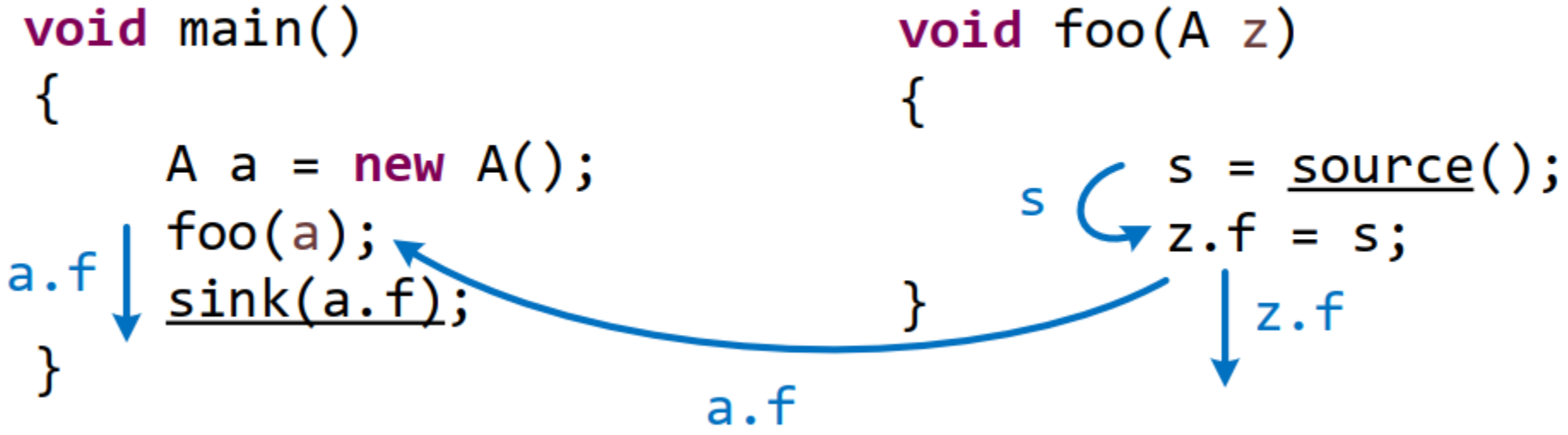
PADERBORN UNIVERSITY

Fraunhofer IEM

Johannes Späth

Fraunhofer IEM

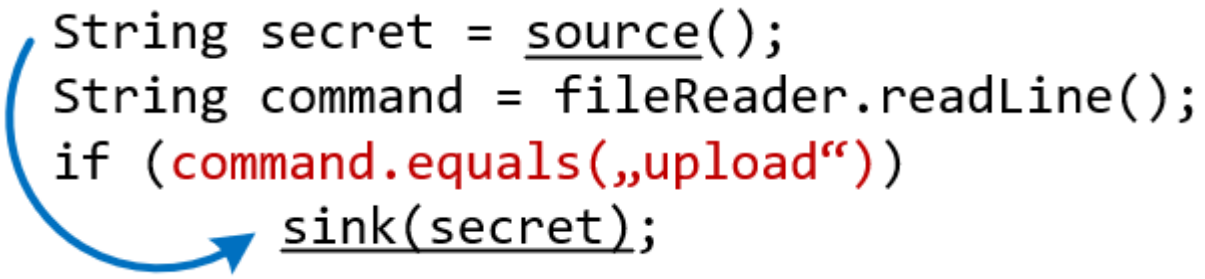
Static taint analysis for detecting data leaks



Flow-sensitive Field-sensitive

Context-sensitive Object-sensitive

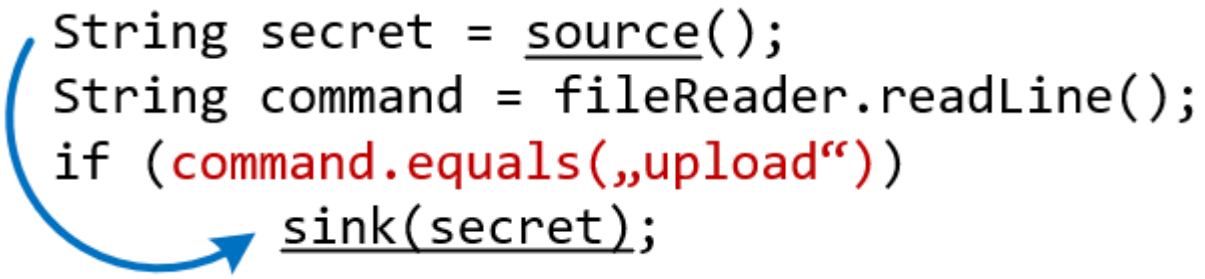
State-of-the-art taint analyses are not path-sensitive



Static taint analysis tools:

- **Will tell**: there is a *taint flow* between source and sink – a *potential* leak
- **Won‘t tell**:
  - which value does secret hold at runtime
  - this taint flow can only happen when the command is „upload“.

Using dynamic analysis for validating static findings



Dynamic analysis tools must run a test case:

- Covering the execution path
- In this example a file contains a String „upload“

Hard to generate such a test case!

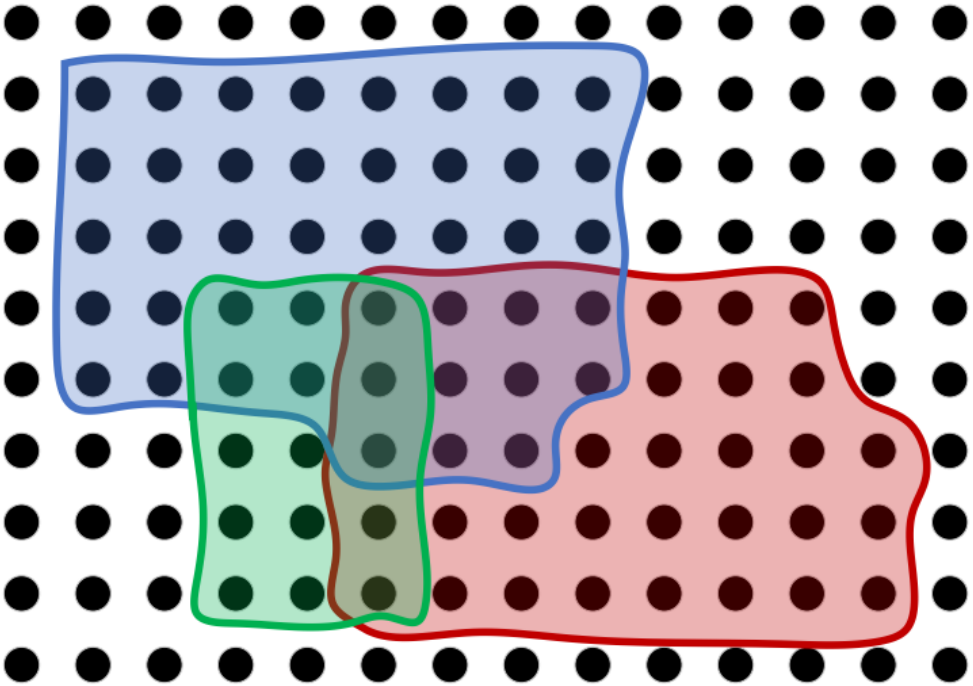
```
String secret = null;

if (condition 1)
    if (condition 2)
        if (condition 3)
                ...
                    if(condition M)
                            secret = source();

if (condition M+1)
    if (condition M+2)
        if (condition M+3)
                ...
                    if (condition M+N)
                            sink(secret);
```

Research Goal

- To study Android taint-analysis results on real world apps
- To understand what kind of path conditions are relevant
- How taint flows are conditioned on different factors



● Taint flow

Factor 1
Factor 2
Factor 3
...

We seek to identify …

I/O operations

```
String secret = source();
String command = fileReader.readLine();
if (command.equals(„upload“))
        sink(secret);
```

UI interactions

```
String secret = source();
button.setOnClickListener(new OnClickListener(){
   void onClick(View view){
        sink(secret);
   }
});
```

Configurations

```
String secret = source();
if (VERSION.SDK_INT < 20)
        sink(secret);
```

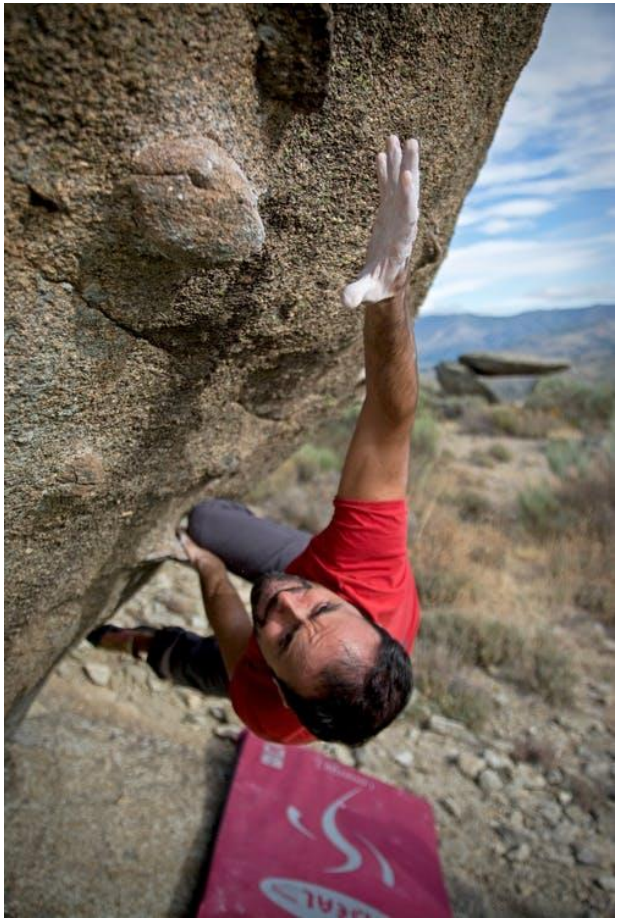
Research Questions

What types of taint flows does FlowDroid report?
How common is each type?

FlowDroid

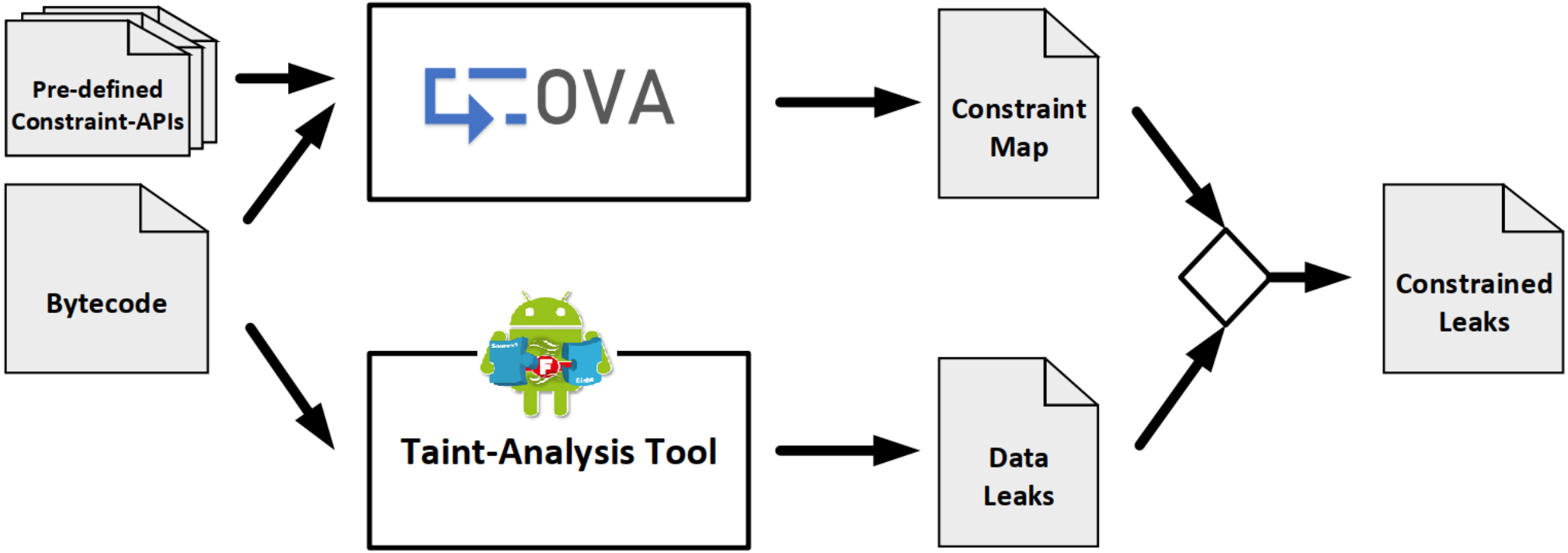
Challenges

Can‘t simply make static taint analysis path-sensitive:



- analysis may not scale
- analysis of path constraints is a non-distributive problem
  - FlowDroid built on top of IFDS
  - IFDS solves distributive problems

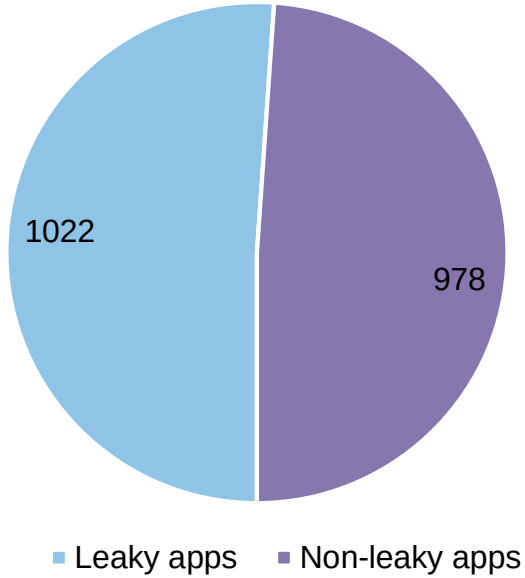
Our Solution



Methodology

- Dataset: 2000 Android apps (2016-2018) from AndroZoo
- FlowDroid v2.5.1 in default configuration
- We collected constraint-APIs for COVA
  - 335 APIs for UI interactions
  - 448 APIs for Configurations
  - 120 APIs for I/O operations
- Semi-automated study in two steps
  - Filter false positives
  - Classifying taint flows wrt. path constraints

FlowDroid reports 28176 taint flows in 2000 apps



Step 1: Filter false positives

Stratified random sampling taint flows according to top source-sink-pairs

	Group	#Flows	#Apps	#Sampled Apps
Intra-procedural	A	2,193	535	54
	B	1,410	199	20
	C	194	166	17
Inter-procedural	D	1,440	156	16
	E	862	291	30
	F	847	85	9

False-positive patterns found in step 1

Group A & E (source = openConnection, sink = setRequestProperty)

```
HttpURLConnection c = (HttpURLConnection) new URL("http...").openConnection();
c.setDoInput(true);
c.setRequestProperty("User-Agent", "Mozilla/5.0");
```

Group B (source = obtainMessage, sink = sendMessage)

```
Message m = handler.obtainMessage();
handler.sendMessage(m);
```

Inappropriate sources

Takeaways

11/47 default sources used by FlowDroid are inappropriate.

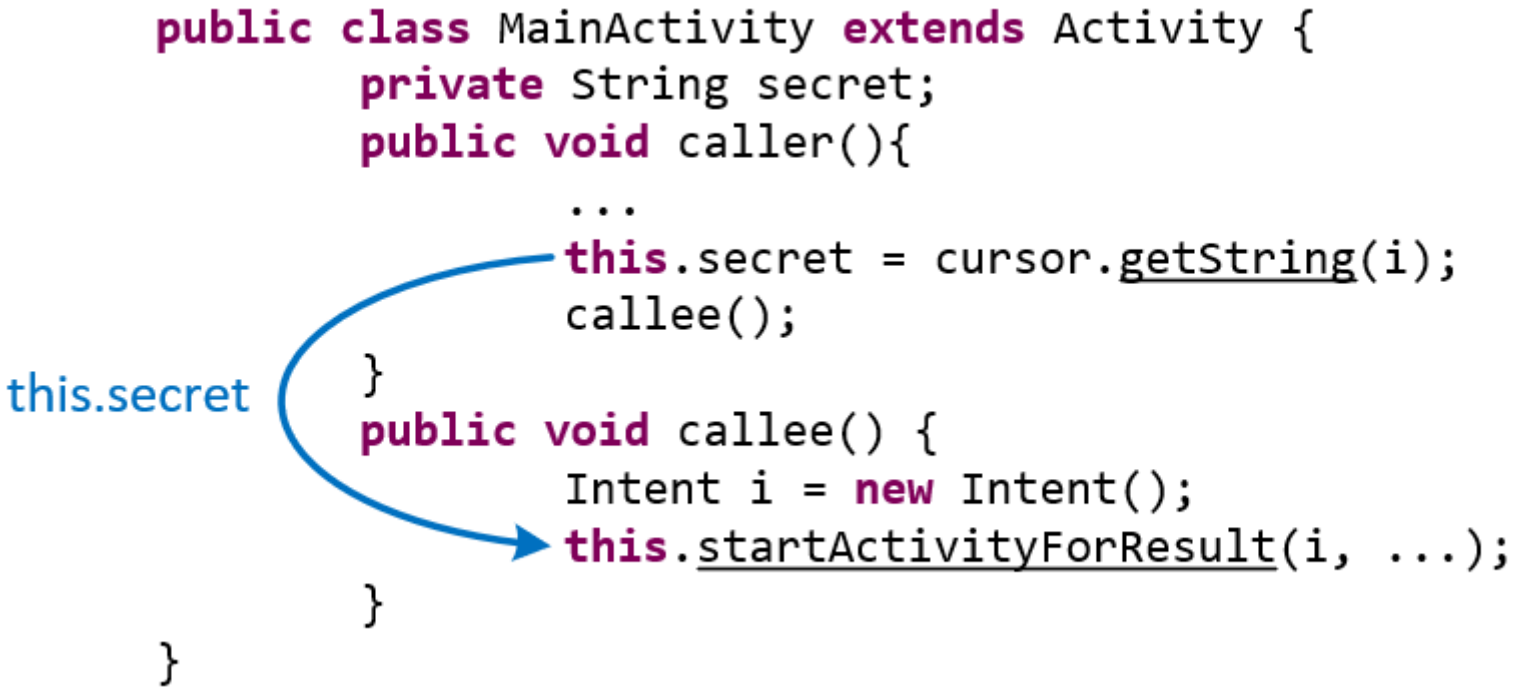
They cause 28% of taint flows being false positives.

Researchers who used FlowDroid in the default configuration may need to re-evaluate their conclusions.

In a short investigation, we found 9 papers in which the work was built on top FlowDroid and inappropriate sources and sinks were used.

False-positive patterns in step 1 (continued)

Group D (source = getString, sink = startActivityForResult)



Takeaways

Taint flows with taints connecting sources and sinks on the same objects are false positives.

For a better precision, such approximated rule should not be used for all sources and sinks, but only in certain cases.

Step 2: Classify taint flows wrt. path constraints

getX() returns a configuration option
getY() returns an input from a file

COVA

```
String secret = null;
boolean x = getX();
if (x)
    secret = source();
boolean y = getY();
if (y)
    sink(secret);
```

X

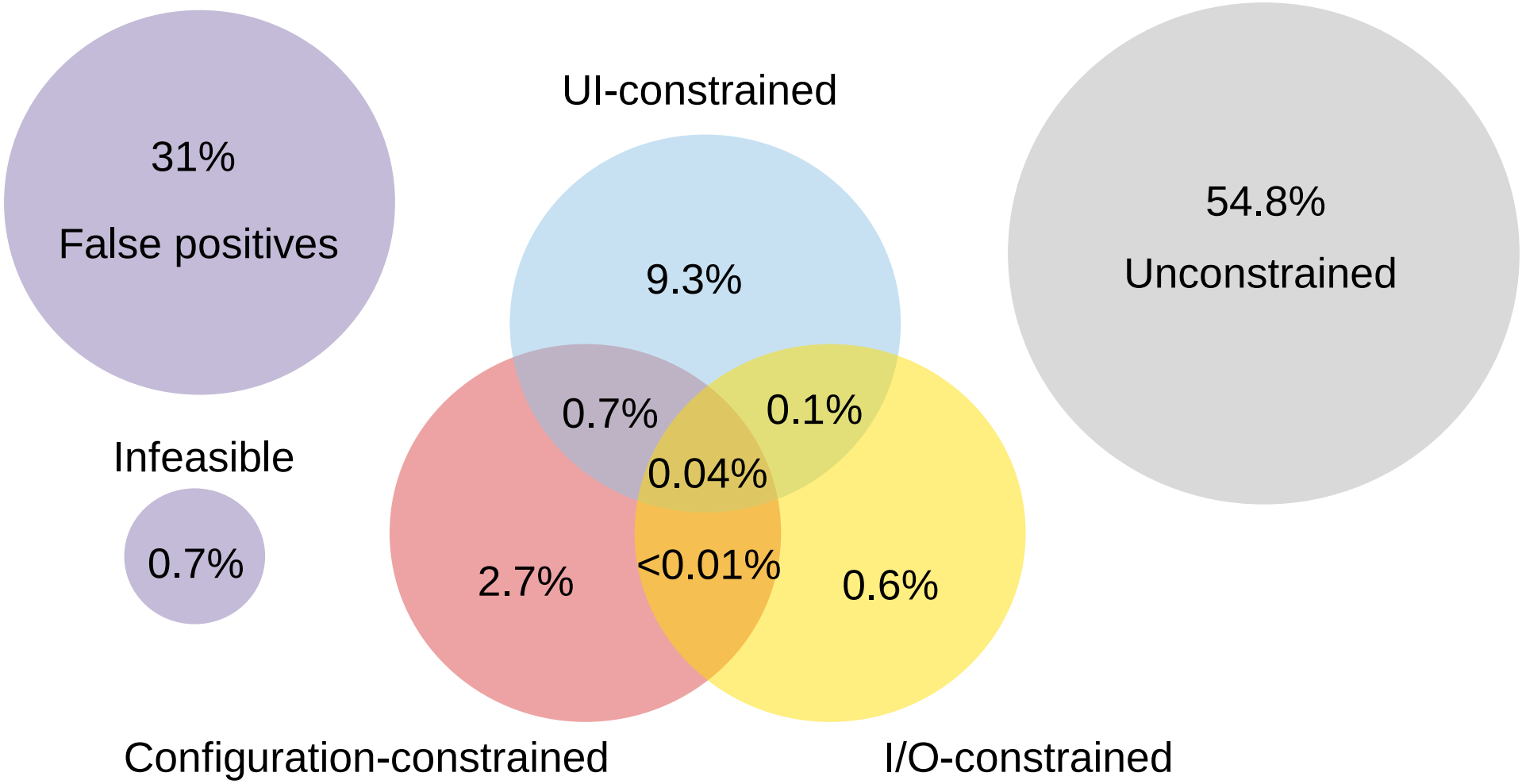
Y

X ∧ Y

Configuration-constrained

IO-constrained

Classification in step 2



Takeaways

Taint flows are seldom conditioned by combinations of the three factors.

Thus, most taint flows could be dynamically confirmed by different tools that specialize on the respective category.

Summary

State-of-the-art taint analyses are not path-sensitive

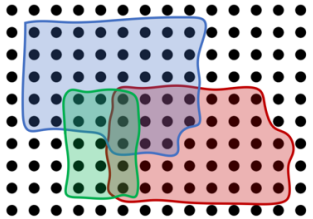
```
String secret = source();
String command = fileReader.readLine();
if (command.equals(„upload“))
    sink(secret);
```

Static taint analysis tools:

- **Will tell**: there is a *taint flow* between source and sink – a *potential* leak
- **Won‘t tell**:
  - which value does secret hold at runtime
  - this taint flow can only happen when the command is „upload“.

Research Goal

- To study Android taint-analysis results on real world apps
- To understand what kind of path conditions are relevant
- How taint flows are conditioned on different factors



● Taint flow

Factor 1
Factor 2
Factor 3
…

We seek to identify …

I/O operations

```
String secret = source();
String command = fileReader.readLine();
if (command.equals(„upload“))
    sink(secret);
```

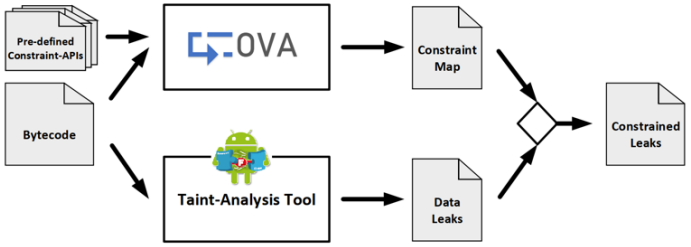
UI interactions

```
String secret = source();
button.setOnClickListener(new OnClickListener(){
    void onClick(View view){
        sink(secret);
    }
});
```

Configurations

```
String secret = source();
if (VERSION.SDK_INT < 20)
    sink(secret);
```

Our Solution



False-positive patterns found in step 1

Group A & E (source = openConnection, sink = setRequestProperty)

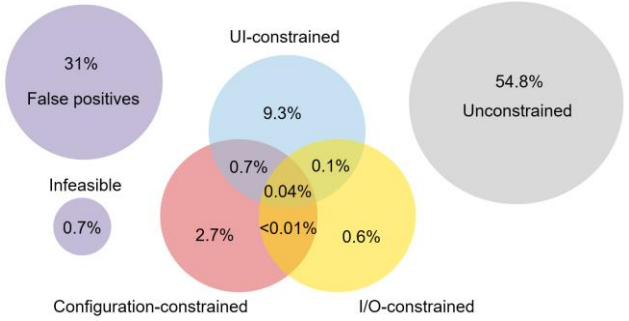
```
HttpURLConnection c = (HttpURLConnection) new URL("http...").openConnection();
c.setDoInput(true);
c.setRequestProperty("User-Agent", "Mozilla/5.0");
```

Group B (source = obtainMessage, sink = sendMessage)

```
Message m = handler.obtainMessage();
handler.sendMessage(m);
```

Inappropriate sources

Classification in step 2



https://github.com/secure-software-engineering/COVA

COVA

COVA: A Static Analysis Tool for Computing Partial Path Constraints

Constraint MAP	Code
TRUE	1 String secret = source();
TRUE	2 button.setOnClickListener(new OnClickListener(){
TRUE	3 void onClick(View view){
CLICK	4 int sdk = Build.VERSION.SDK_INT;
CLICK	5 if (sdk < 20)
CLICK ∧ SDK<20	6 sink(secret);
CLICK ∧ SDK≥20	7 else
CLICK ∧ SDK≥20	8 doSth1();
CLICK ∧ SDK≥20	9 if (sdk < 20)
FALSE	10 doSth2();

https://github.com/secure-software-engineering/COVA

COVA

Feel free to contact me if you have any questions

linghui@outlook.de