A Qualitative Analysis of Taint-Analysis Results

HEINZ NIXDORF INSTITUT
UNIVERSITÄT PADERBORN

Linghui Luo (Paderborn University), **Eric Bodden** (Paderborn University&Fraunhofer IEM) and **Johannes Späth** (Fraunhofer IEM)

linghui.luo@uni-paderborn.de @LinghuiLuo https://linghuiluo.github.io/

Motivation

Problem:

- Static analysis tools implement only may-analysis.
- Reported warnings are inconclusive.
- Developers are forced to identify and prioritize excessive warnings manually.

Observation:

There are different reasons for may-analysis results.

Reason 1 – Program Input

```
String secret = source();
String input = bufferedReader.readLine();
if(input.contains("address")){
        print(secret); // depends on input
}
```

Example 1: Definite data leak under certain user inputs

Reason 2 – Environment Configuration

```
String secret = source();
boolean a = confi.getOption("A").isOn();
if(a){
        print(secret); // depends on option A
}
```

Example 2: Definite data leak under certain configuration

Reason 3 – Analysis Limitation

```
String secret = source();
if(isPrimeNumber(5754853343)){
        print(secret); // branch uncertain
}
```

Example 3: Possible data leak

Goal

Hypothesis:

Warnings reported by static analysis tools can be categorized according to different kinds of reasons (path conditions). Based on the path conditions, warnings can be prioritized directly or validated dynamically.

Study Object: Leaks reported by FlowDroid

Research Questions:

RQ1: Do „low-hanging fruits" exist, and if so, what characteristics do these leaks have?

RQ2: What types of leaks does FlowDroid report?

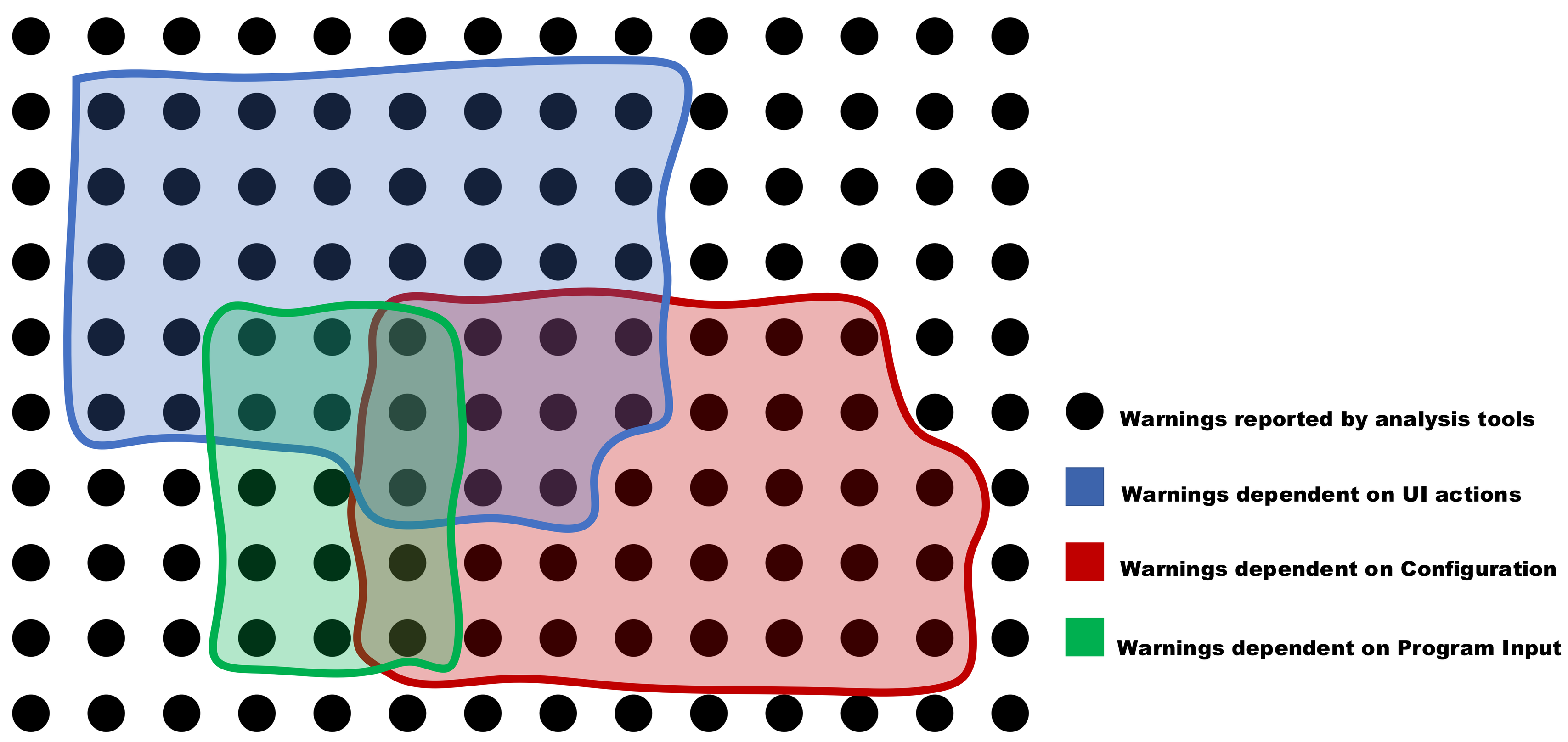


Figure 1. Categorization of warnings reported by static analysis tools

Approach

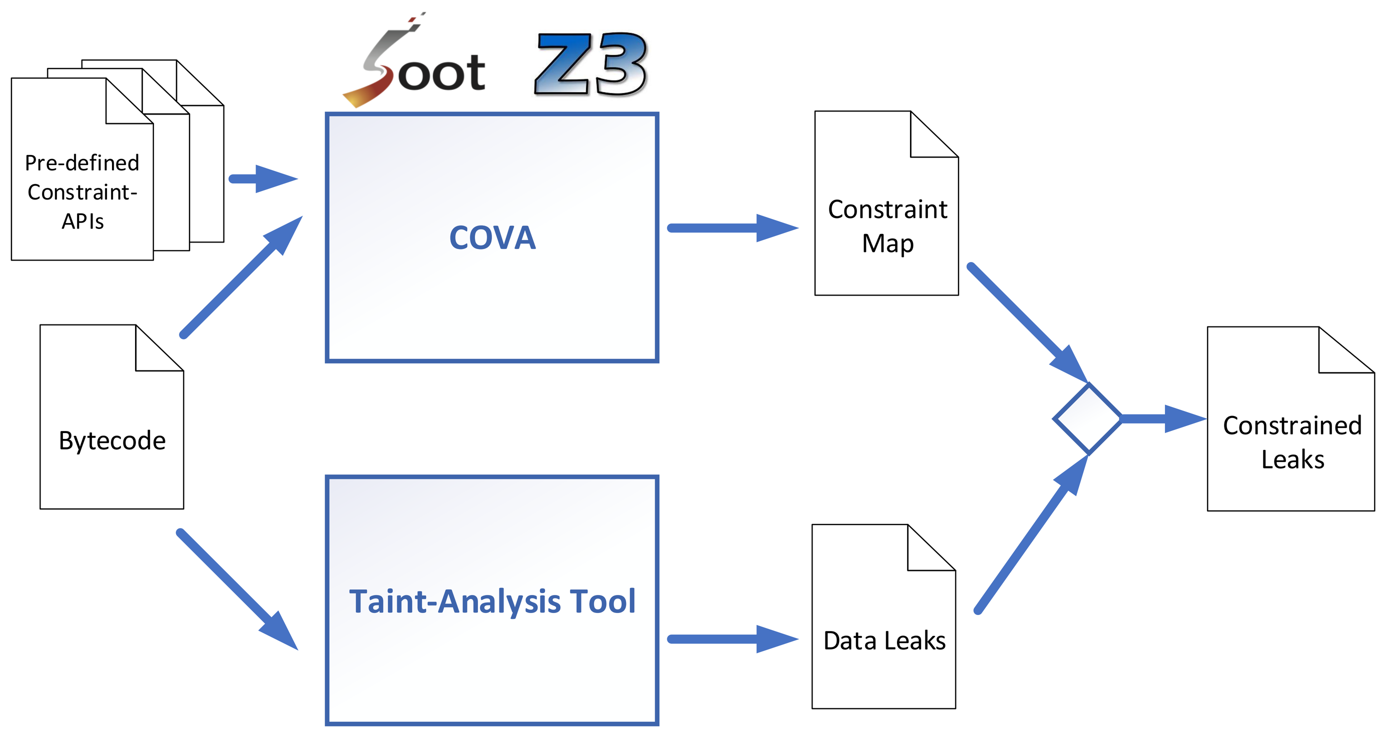


Figure 2. Workflow of applying COVA to taint-analysis results

```
public class LeakyApp extends Activity {
  private TelephonyManager tm;
  @Override
  protected void onCreate(Bundle bundle) {
    ...
    button.setOnClickListener(new View.OnClickListener() {
      @Override
      public void onClick(View view) {
        String model= Build.MODEL;
        if (model.equals("HTC Desire")){
          String deviceID= tm.getDeviceId();
          send(deviceID);
        }
    });
  }
  private void send(String deviceID){
    int sdk = Build.VERSION.SDK_INT;
    if (sdk < 25) {
      SmsManager sms = SmsManager.getDefault();
      sms.sendTextMessage("+4912", null, deviceID, null, null);
    }
  }
}
```

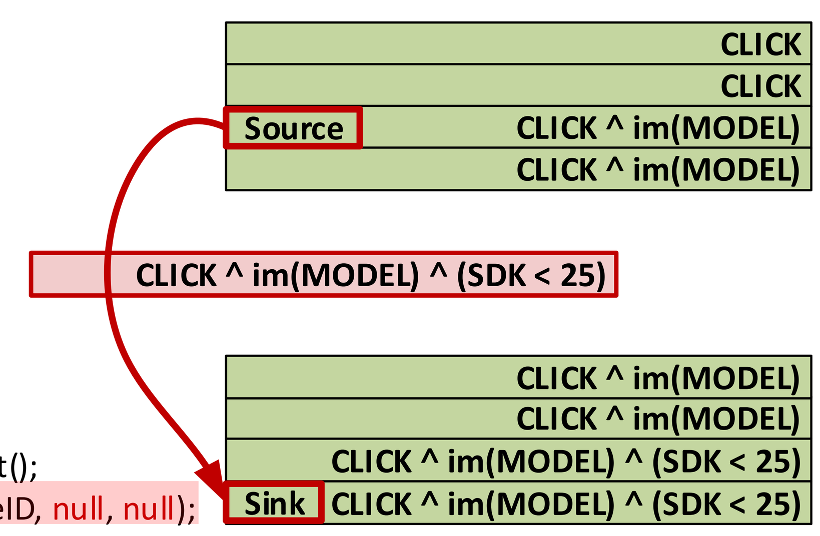


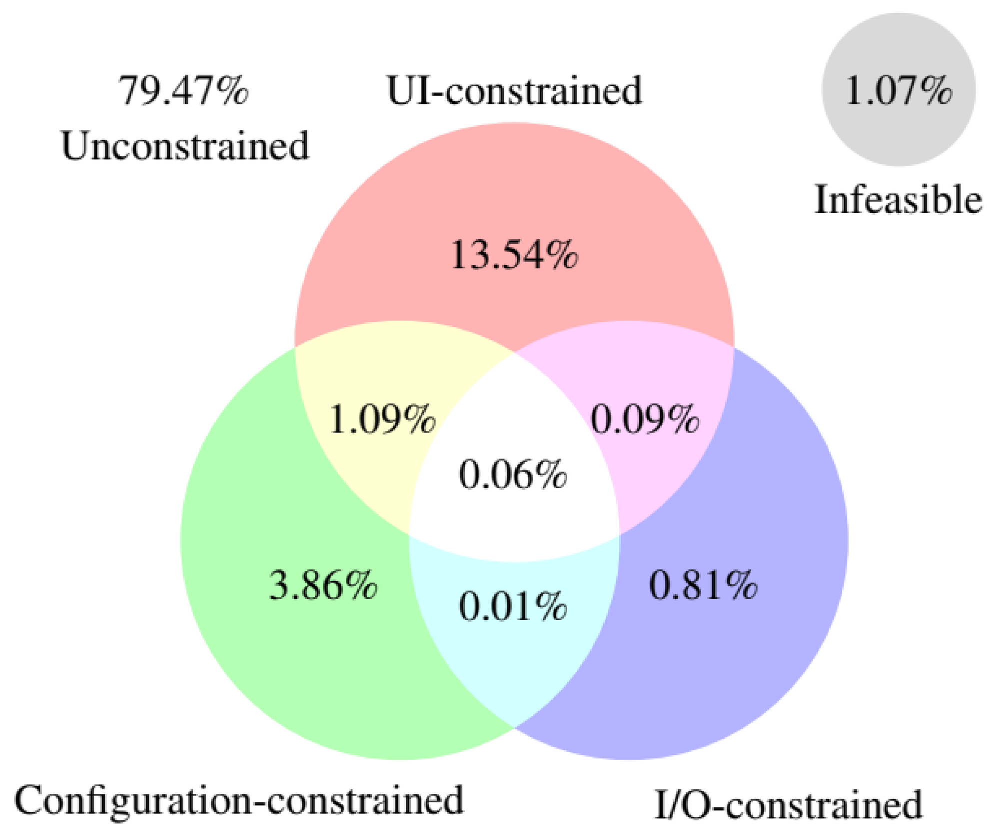
Figure 3. Example of reasoning reported data leak

Results

Empirical Study:

- Implemented a static analysis tool called COVA which computes path constraints. It is based on Soot, VASCO, Z3 and Boomerang, supporting analysis for Java and Android applications.
- Conducted a COVA-supported qualitative study of leaks reported by FlowDroid from 1,022 real-world Android apps.

Figure 4. Different types of leaks



- **UI-constrained:** leaks are conditioned on user interactions.
- **Configuration-constrained:** leaks are conditioned on environment settings such as platform version, country, etc.
- **I/O-constrained:** leaks conditioned on specific data inputs via I/O streams or file system
- **Infeasible:** leaks with infeasible path constraints.

Graduiertenkolleg "Human Centered Systems Security"
NERD.NRW

SOFTWARE ENGINEERING GROUP