EDGEMINER: AUTOMATICALLY DETECTING IMPLICIT CONTROL FLOW TRANSITIONS THROUGH THE ANDROID FRAMEWORK

Yinzhi Cao, Yanick Fratantonio, Antonio Bianchi, Manuel Egele, Christopher Kruegel, Giovanni Vigna, and Yan Chen

Columbia, UCSB, BU, NU

Introduction

- Static analysis has been used for security and privacy.
- Many analyses rely on the control flow graph.
- Challenge in Android: the framework is 8.6 million lines of code
- Ignoring the framework -> incorrect control flow graph of Android apps
  - Common cause for imprecision: “callbacks”, e.g., onClick

A Motivating Example

```
public class MainClass {
    static String url;
    public static void main(String[] args) {
            MalComparator mal = new MalComparator();
            MainClass.value = 42
            Collections.sort(list, mal);
            sendToInternet(MainClass.value);
    }
}
public class MalComparator implements Comparable<Object> {
    public int compare(Object arg0, Object arg1) {
        MainClass.value = GPSCoords
        return 0;
    }
}
```

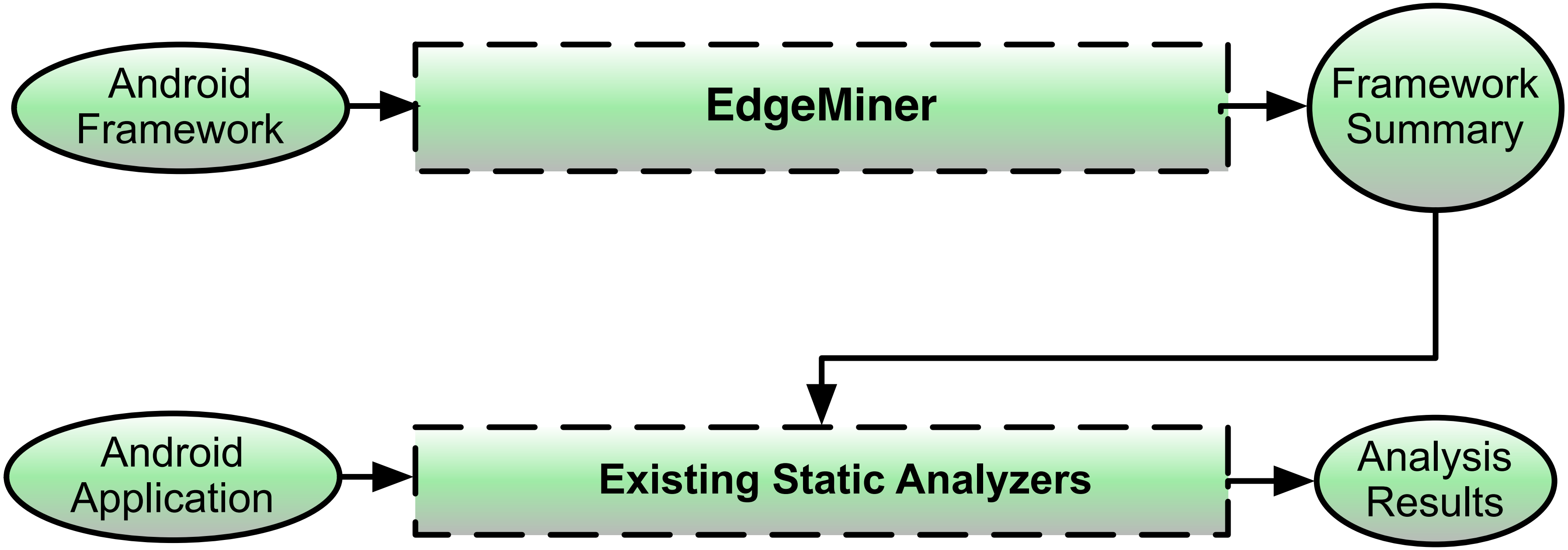
Privacy leakage is up to the value of MainClass.value.

Existing Approaches

- Whole program analysis
  - State explosion
  - Pushing existing static analysis systems to their limits
  - Redundant Efforts (slow-down of static analysis)
- Summary-based analysis
  - Manual summarization: impossible due to the high volume of callbacks
  - Heuristic summarization: inaccurate

EdgeMiner: Usage Scenario

- Summarize framework: list of registration-callback pairs

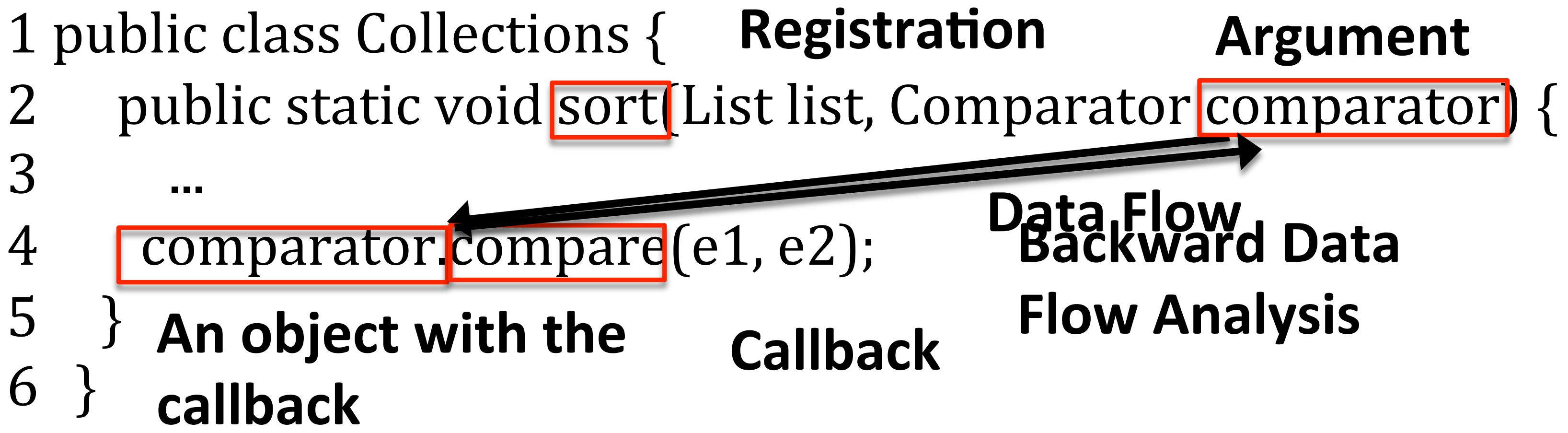


Concepts

- Callback
  - Necessary condition: a framework method that can be overridden by an application method

- Registration
  - Necessary condition: a framework method that is invokable from the application space

A Data Flow



```
public class Collections {
    public static void sort(List list, Comparator comparator) {
        ...
        comparator.compare(e1, e2);
    }
}
```

Registration

Argument

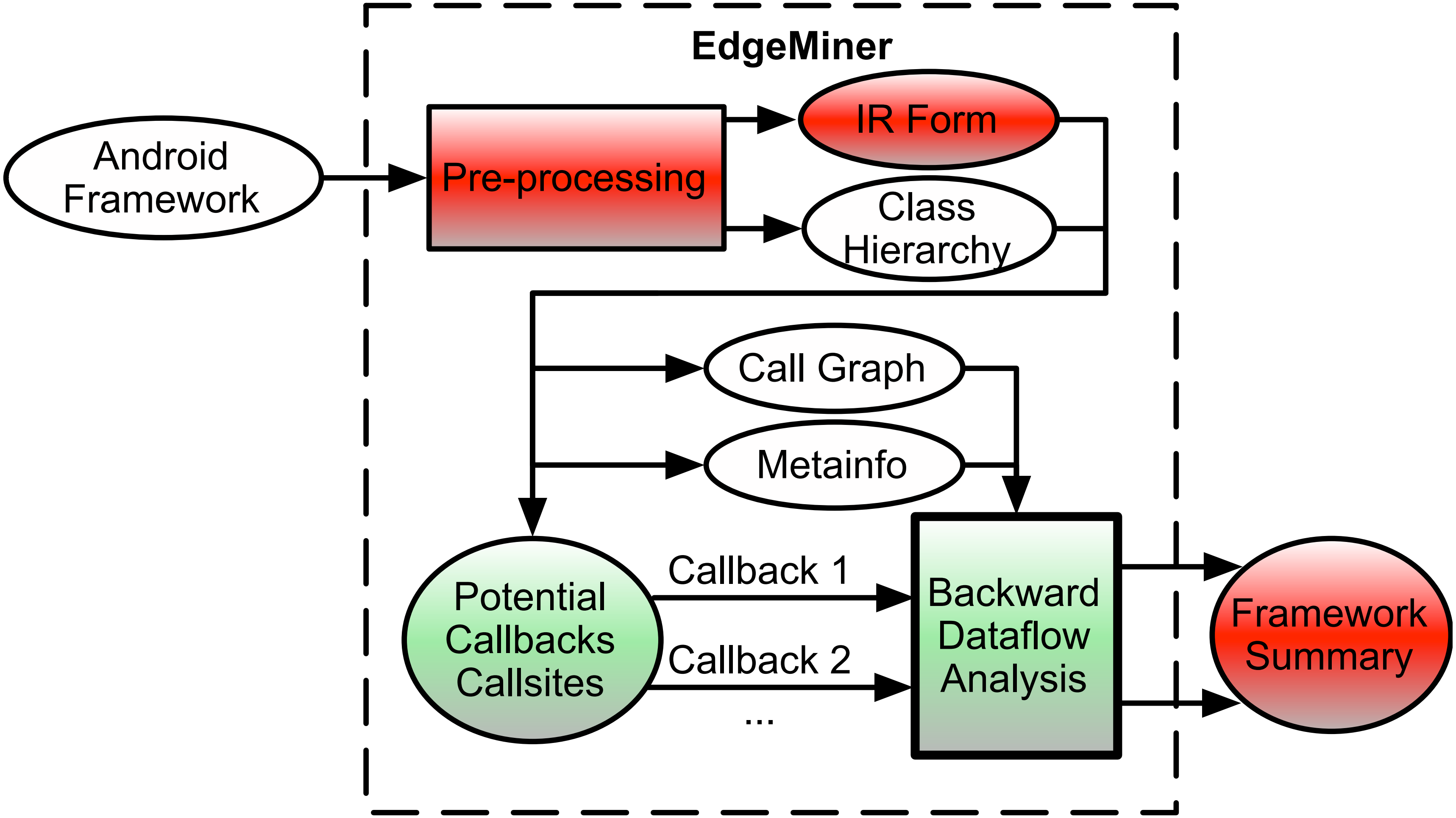
Data Flow

Backward Data Flow Analysis

An object with the callback

Callback

System Architecture



Implementation

- ROP intermediate representation (IR)
  - Well-suited for static analysis
  - In SSA form
  - Integral part of Android SDK
- EdgeMiner
  - Built on top of ROP
  - Performs backward dataflow analysis
  - Summarizes implicit control flows through framework

Results

□ Number of registrations and callbacks

Android Version	# Registrations	# Callbacks	# Pairs
2.3 (API 10)	10,998	11,044	1,926,543
3.0 (API 11)	12,019	13,391	2,606,763
4.2 (API 17)	21,388	19,647	5,125,472

□ Results for Android 4.2 at

http://edgeminer.org

Accuracy

- False negative
  - Compare with dynamic approach
    - Incomplete but accurate
  - 8,195 randomly selected applications
  - 6,906 registration-callback pairs
  - EdgeMiner finds all pairs
- False positive
  - Manual inspection
  - Eight false positives out of 200 pairs

Improving FlowDroid

- Integration with FlowDroid
  - Synchronous callbacks: inline invocation
    - E.g., Collections.sort and Comparator.compare
  - Asynchronous callbacks: delayed invocation
    - E.g., setOnClickListener and onClick

Pattern	# FlowDroid	# EdgeMiner
Listener	155	576
Callback	19	319
On	3	509
None of the above	4	18,243
Total	181	19,647

Improving FlowDroid – Accuracy

Tool	FlowDroid	FlowDroid + EdgeMiner
# Apps with ≥ 1 privacy leak	285	294 (285 + 9)
# Privacy leaks (in total)	46,586	51,418
# Apps timeout	48	48

- Run 9 new apps in TaintDroid
  - 4 verified, 2 crash, and 3 no leak
- Incorrect call graph -> missed privacy leaks
- Performance
  - 34.7 seconds one-time loading
  - Only 1.85% overhead added to FlowDroid

Conclusion

- EdgeMiner summarizes implicit control flows in Android framework
- EdgeMiner identifies registration-callback pairs through backward data flow analysis
- Results improve state-of-the-art static Android analyses
  - FlowDroid detected 9 additional apps with leaks

Thank you!

Questions?

Results are available at

http://www.edgeminer.org