Abstract

We present a precise, path-sensitive static analysis for reasoning about heap reachability; that is, whether an object can be reached from another variable or object via pointer dereferences. Precise reachability information is useful for a number of clients, including static detection of a class of Android memory leaks. For this client, we found the heap reachability information computed by a state-of-the-art points-to analysis to be too imprecise, leading to numerous false-positive leak reports. Our analysis combines a symbolic execution capability of path-sensitivity and strong updates with abstract heap information computed by an initial flow-insensitive points-to analysis. This novel mixed representation allows us to achieve both precision and scalability by leveraging the initial points-to-analysis result to guide execution and prune infeasible paths. We have implemented our techniques in the Thresher tool, which we have used to find several developer-confirmed leaks in open-source Android applications.

Causes of Android Activity Leaks

Phone rotation (1) triggers Activity destruction (2) and re-creation (3).

Why Does Points-To Analysis Fail?

Standard flow-insensitive points-to analysis cannot handle the use of the null object pattern in our example: we need path-sensitivity, flow-sensitivity, context-sensitivity, and strong updates to reason about the relationship between EMPTY, size, and capacity.

```java
public class Main {
    public static void main(String[] args) {
        Act a = new act0.Act(); a.onCreate();
    }
}
```

```
null object pattern: tbl initially points to shared EMPTY array to save overhead of allocating array for each instance created
```