1 Background

- Fuzzing is a method to find vulnerabilities
  - It finds a lot of crashes automatically by generating inputs
- Root cause analysis is difficult
  - Fuzzers only report crashes in location
  - Root cause is often far from crash location

2 Previous Root Cause Analysis

- Symbolic execution [1]
  - Collect conditions to trigger crashes during execution
  - Path explosion problem 😡
- Statistical Crash Analysis [2]
  - Compare behavior of predicates using similar inputs
  - 🙄 Binary-level information and high false positive rate 😡

3 Human friendly data flow graph (DFG)

Our goal:
Quickly and easily understand root cause

Key ideas:
(a) Difference of DFG between crashed and normal execution
(b) Adding lexical information at the source code level

4 Implementation

- Lexical information extraction: source code instrumentation using LLVM
- Data flow tracing: created ourselves from scratch

5 Preliminary experiments

Evaluation dataset:
- Magma [3]: Reproduces known software vulnerabilities
  - Provides (non-)crash inputs as artifacts

Evaluation method:
- Measure distance between data flow difference and patch point

How to read graph:
- Compare data flows of (non-)crash inputs, and highlight nodes which (dis)appear in other graph
- Type: (ObjType) type: (Object *)
- (Omitted) Data flow difference

Result (4 cases):
- (Distance 0) Case 1, 2: Patch point was control flow governing the boundary of difference
- (Distance 0) Case 3: Patch point was tangent to the boundary
- (Distance 1) Case 4: A sanitization patch was applied to a point at distance 1 from the boundary

6 Future work

- Evaluate our approach in more crash cases
- Evaluate whether our approach eases root cause analysis compared to manual analysis and previous studies

References