Taming a Fuzzer Using Delta Debugging Trails

*Yuanli Pei, Arpit Christi, Xiaoli Fern, Alex Groce and Weng-Keen Wong*

School of EECS
Oregon State University
Fuzzing (Random Testing)
Fuzzing (Random Testing)

- Applications:
  - C Compilers -- *Csmith* [Yang et al, 2011]
    - more than 450 bugs
  - Javascript engine -- *jsfunfuzz* [Ruderman, 2007a, 2007b]
    - more than 1700 bugs
  - flight software for the Curiosity Mars Rover mission [Groce et al, 2014]
Issues with Fuzzers

- **Testing and Debugging Dilemma**
  - Testing: more effective at finding bugs when test cases are large [Andrews, 2008]
  - Debugging: large test cases are generally difficult to debug

- **Indistinguishability of bugs**
  - Fuzzers often generate many test cases triggering the same bug

*Figure 1. A fuzzer tends to hit some bugs thousands of times more frequently than others [Chen et al, 2013]*
Address Issues of Fuzzers

• **Testing and Debugging Dilemma**
  • Solution: Reduce large test cases (e.g., delta debugging)

• **Indistinguishability of bugs**
  • Solution: Fuzzer Taming (focus of this work)

  Given a large collection of test cases, each of which triggers a bug, find a ranking $\pi$ such that more distinct bugs are triggered early in the list.
Previous Work [Chen et al, 2013]
Reducing: Delta Debugging trials

Original

Test case trial \( X_i \)

System (e.g., compilers)

fail \( \times \)

System (e.g., compilers)

fail \( \times \)

System (e.g., compilers)

fail \( \times \)

\vdots

System (e.g., compilers)

fail \( \times \)

System (e.g., compilers)

pass \( \checkmark \)

Minimal reduced \( X_{i1} \)
This Work

- Reducing: Delta debugging trials
- Taming: Furthest Point First Ranking
Reducing: Delta Debugging trials

• Each test case \( X_i = [x_{i1}, x_{i2}, \ldots, x_{in_i}] \)

• Keep the minimal reduced \( X_{i1} \)

• Policy for non-minimal test cases:
  (a) Number of additional test cases (at most \( M \) additional test cases)
  (b) Closeness to minimal test case
  (c) Giving up early
Taming: Furthest Point First

• Set of top-k ranked test cases

\[ Q^k = \{ X_{\pi(1)}, \ldots, X_{\pi(k)} \} \]

• FPF ranking criterion

\[ X_{\pi(k+1)} = \arg\max_{X_i \in \mathcal{X} \setminus Q^k} \left\{ \min_{X_j \in Q^k} d(X_i, X_j) \right\} \]
Taming: Furthest Point First

- Distances
  - Single linkage
    
    \[ d(X_i, X_j) = \min_{1 \leq u \leq n_i, 1 \leq v \leq n_j} d(x_{iu}, x_{jv}) \]
  
  - Average linkage
    
    \[ d(X_i, X_j) = \frac{1}{n_i \cdot n_j} \sum_{1 \leq u \leq n_i, 1 \leq v \leq n_j} d(x_{iu}, x_{jv}). \]
Taming: Furthest Point First

• Justification

• Assumption: larger \( d(X_i, X_j) \) \( \Rightarrow \) larger probability that \( X_i \) and \( X_j \) trigger different bugs

• FPF is optimal since it maximizes the minimum pairwise distances for every top-k list!
Experiments

• Data
  • Software: Javascript engine
  • Fuzzer: jsfunfuzz
  • Features of test cases:
    • lexical features: histogram of tokens (meaningful character strings)
    • function coverage features: number of times the engine called the i-function while executing the test case

• Baselines
  • Random ordering
  • Reduced [Chen et al, 2013]: minimal reduced + FPF ordering
Experiments

(a) Test case features, first 150 tests

(b) Test case features, all tests
Experiments

(c) Function coverage features, first 150 tests

(d) Function coverage features, all tests
Conclusions

- Delta debugging trails help improve the early part of the FPF ranking
- Lexical features from test cases are more effective
References


