Applying Fuzzing Metrics

Dr. Jared DeMott
CTO AND FOUNDER

@jareddemott www.vdalabs.com

Overview

Course summary

When are we done?

Did we find all the bugs?

Course Summary

Traditional fuzzers

- Mutation
  - Easy
- Generation
  - ROI

Niche

- API, In-memory

Add

- Monitoring
- Distribution
- Code coverage
  - Gcov or the like

Course Summary

Next-gen fuzzers

- Feedback
  - Pros
    - Typically include monitoring, coverage, and distribution
    - Better guarantees around bugs
  - Cons
    - Runs slower than generation
    - Harder to setup
      - Local process with single input

Metrics

- Adoption
- Bugs
- Tools
- Time
- Coverage
- Measurement

Adoption

Number of people/projects fuzzing

Bugs

Count by

- Tool
- Technique
- Team
- Time
- etc.

Tools

Diversity is good

- Different fuzzers find different bugs
  - In-house vs. public
  - Custom vs. framework
- Mutation
  - Quicker to setup, but may yield less unique results
- Generation
  - Better results, but is more expensive
- Feedback
  - Best if applicable

Time

Clock

- How long

Iterations

- How many

When is enough, enough?

- Bug bars

Coverage

Code

- Branch coverage
- Path coverage

System

Measurement

Learners take tests to measure retention

Organizations can test readiness

Improvement

- More of X and less of Y than last quarter
- Pentests
  - This is a bit tricky to do well

Policy

All groups with native code

A number of iterations

B tools

C coverage

D improvements

Fuzzing Misconceptions

I'll give you the hardware if you start a fuzzer real quick for me

- Fuzzing is so much more than hardware

Fuzzing is better than X or Y

- They all fit together

Summary

Make code better through fuzzing

- Variety of tools and techniques
- No one size fits all
- In-house expertise is needed
- Measure progress