Fuzzing

History

- Miller et al. "An Empirical Study of the Reliability of UNIX Utilities", CACM 33(12), 1990

Why do we care?

- Pwnie award (2009)

Fuzzing

- Generate random input
  - Command line args, files, network tr function argum
- Run program
- Check for errors

- Low probability of hitting a bug
  - Can we do better?

Mutational Fuzzing

- A.K.A. dumb fuzzing or black-box fuzzing
- Randomly change ***seed*** input
- Example
  - Standard HTTP
    - `GET /index.h`
  - Mutated requests
    - `AAAAAA...AAAA /index.html`
    - `GET ///////index.html HTTP`
    - `GET %n%n%n%n%n%n.html HTTP/1.1`
    - `GET /AAAAAAAAAAAAA.html HTTP/1.1`
    - `GET /index.html HTTTTTTTTTTTTTP/1.1`
    - `GET /index.html HTTP/1.1.1.1.1.1.1.1`

Mutational Fuzzing

- Little or no knowledge of target input format required
  - Do need some valid input as seeds
- Easy to set-up
- Mutation heuri
  - Purely random
  - Flip a bit (or a group of bits)
  - Change byte values
  - Known integers
  - Input splicing
  - Insert or delete bytes

Fuzzing a PDF viewer

Credit: Tal Garfinkel – Stanford/VMware

- Google for .pdf (about 4 billion results)
- Crawl pages to build a corpus
- Use fuzzing too
  - Grab a file
  - Mutate that file
  - Feed it to the program
  - Record if it crashed (and input that crashed it)

Mutational Fuzzing – cons

- Quality depends on choice of seed files
  - One vacation photo is good. 200 is a waste of time.
- Problem with input sanity
  - CRC
  - Compressed fil
  - Language syntax

Generational fuzzing

- A.K.A. grammar-based fuzzing, white-box fuzzing, smart fuzzing
- Creates random input files based on known structure
- Complex to cr
- Require knowledge of the p

Targeting known bugs

- Insert long strings
- Large/small/boundary numbers
- Unicode
- Format strings

How much is enough?

- When to stop? Fuzzing can continue generating random inputs indefinitely
- Example:
  - 300KB input im
  - One specific by
  - Success probability: 0.0000001
  - At 2 seconds per test we expect a crash after a year

Code Coverage

- A metric of how well a code is tested
  - Variety of profiling tools (e.g. `gcov`)

- Three types:

  - Line coverage – een executed
  - Branch coverage – which branc n taken

  - Path coverage – which paths were taken

Code coverage example

```
if (a>2)
      a = 2;
if (b > 2)
      b = 2;
```

- How many test cases are req ull coverage?
- Path coverage: 1
- Branch coverage: 2
- Path coverage: 3

Issues with code coverage

- Does not guarantee no bugs

```
       mySa
  src){
          if
             strcpy(dst, sr
        }
```

- Some parts of the program are unlikely to be covered
  - Error checking, dead code, etc.

More issues with code coverage

- Path explosion
  - n branches require $2n$ test cases for branch coverage, 2^n for code coverage
  - Loops can imply [illegible] aths

- Infeasible paths

```
if (a > 2)
    a = 2;
if (a < 0)
    a = 0;
```

Evolutionary fuzzing

- A.K.A. gray-box fuzzing
- Generates new inputs based on the response from the program
  - Use code cover
  - Prioritise based nctions

Fuzzing problems

- Crashes vs. vulnerabilities
  - A single vulnerability can cause multiple crashes
  - Can hit the same boring bug on many inputs
  - What about bug h?

- Cost
  - Many tests
  - Potentially resource intensive

Chromium ClusterFuzz

- Hundreds of virtual machines
- Thousands of simultaneous instances
- 50,000,000 tests per day (2012 data)
- 10 times bigger
  - 14,366,459,772 test inputs over
  - 112 bugs found