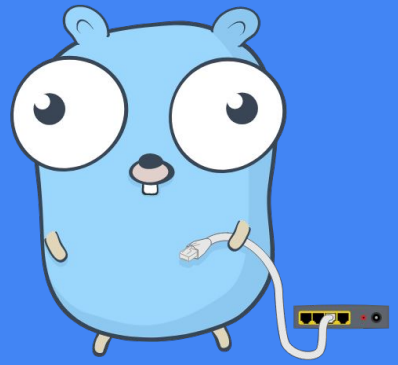
Fuzz Testing



Fuzz Testing aka Fuzzing

“A software testing technique, which basically consists in finding implementation bugs using malformed/semi-malformed data injection in an automated fashion„

“The art of automatic bug finding, and it’s role is to find software implementation faults, and identify them if possible„

History

- 1988 - 1990 - developed at the University of Wisconsin Madison by Professor Barton Miller
- command-line and UI fuzzing
- http://www.cs.wisc.edu/~bart/fuzz/

Attack Types

- Combinations of: numbers, chars, metadata, pure binary sequences
  - Structured vs Unstructured
  - Black, White, Gray box

- Random vs Semi-random
  - Generation based vs Mutation based

Fuzzing Types

- **Application** - the attack vector is the I/O (UI, command line, etc.)
- **Protocol** - proxy, forget packets
- **File Format** - malformed samples and opens them sequentially

Advantages

- Test design is very simple, and free of preconception about the system behaviour
- Find bugs that would be missed by a human eye
- Provides an overall picture of the robustness of the target software

Limitations

- Tend to find simple bugs
- When doing black box it is hard to evaluate the impact of found bug
- Programs with complex inputs can require much more work to produce a smart enough fuzzer to get sufficient code coverage

Fuzzing in programming

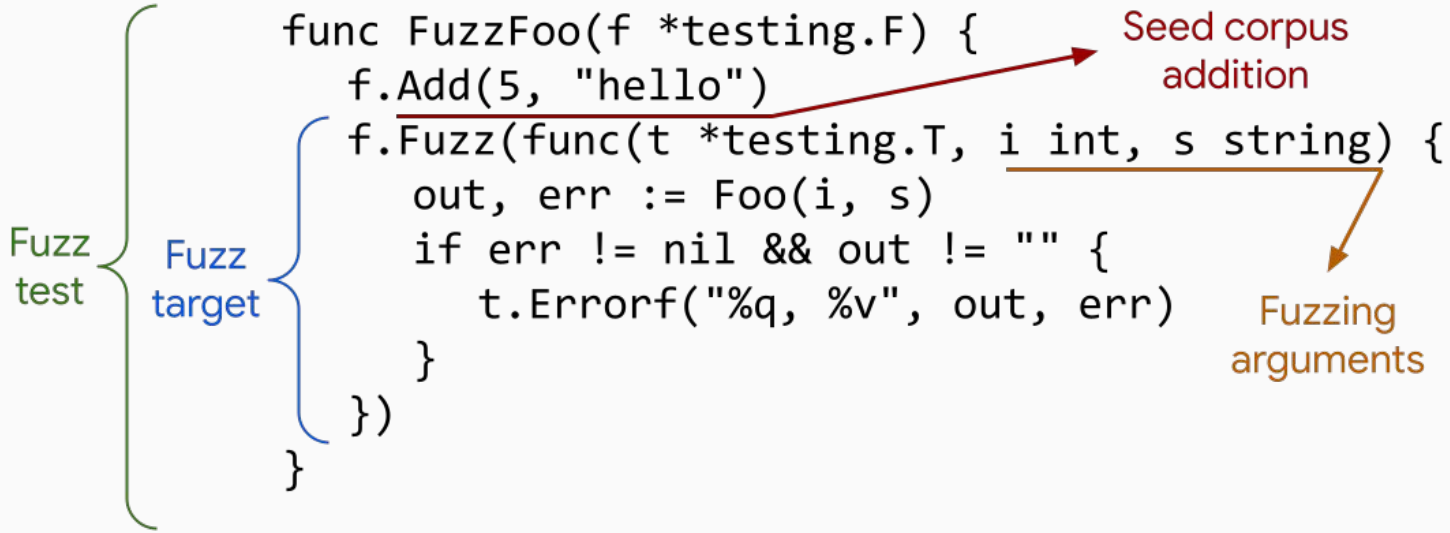
- Great success in C/C++ - Google’s OSS-Fuzz, 50k bugs in 300+ open source projects
  - Memory corruption
- Memory safe languages
  - Useful in discovering other security vulnerabilities
  - Java - Jazzer (integrated into OSS-Fuzz)

Fuzzing in Go

Get Started With Fuzzing in Go

- Motivation: https://github.com/golang/go/wiki/Fuzzing-trophy-case
- Since Go 1.18 (beta)
- Part of the **`testing`** package - no new dependencies needed!
- Declare in **`*_test.go`** files
- Declare as **`func FuzzTestName(f *testing.F) { ... }`**
- Run with **`go test`** (only data from **`/testdata`**)
- Can be run individually as **`go test -fuzz={fuzzTestName}`**

Structure of Fuzz Tests



Output of Fuzz Tests

```
~ go test -fuzz FuzzFoo
fuzz: elapsed: 0s, gathering baseline coverage: 0/192 completed
fuzz: elapsed: 0s, gathering baseline coverage: 192/192 completed, now fuzzing with 8 workers
fuzz: elapsed: 3s, execs: 325017 (108336/sec), new interesting: 11 (total: 202)
fuzz: elapsed: 6s, execs: 680218 (118402/sec), new interesting: 12 (total: 203)
fuzz: elapsed: 9s, execs: 1039901 (119895/sec), new interesting: 19 (total: 210)
fuzz: elapsed: 12s, execs: 1386684 (115594/sec), new interesting: 21 (total: 212)
PASS
ok      foo 12.692s
```

Inner Workings of The Fuzzing System

- **Coordinator** schedules **workers**
- Workers report back to the coordinator
- Workers' main objectives:
  - Extend coverage
  - Find crashing input
- Workers do the heavy lifting
  - Repeatedly mutating & minimizing corpus
  - Running the registered fuzz targets

Corpus Generation & Structure

- Seed corpus
- Corpus generation by mutation
- Corpus minimization
- Corpus caching in **`$GOCACHE/fuzz`**
- On-disk corpus structure:

```
go test fuzz v1
[]byte("hello\\xbd\\xb2=\\xbc ⌘")
int64(572293)
```

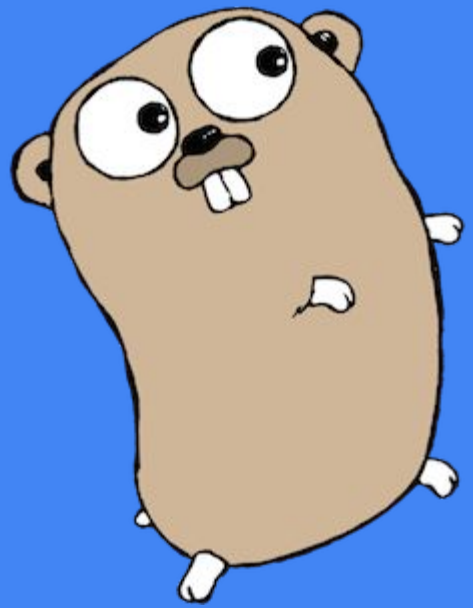
Limitations, Shortcomings & Issues

- Only some types are accepted as fuzzing arguments
  - `string, []byte, int, int8, int16, int32/rune, int64, uint, uint8/byte, uint16, uint32, uint64, float32, float64, bool`
- "Hard" to verify output - no fixed values to check against
- Sub-optimal performance (critique)
- Highest priority issues (from the issue tracker):
  - On-disk corpus not minimized (#49290)
  - Cannot fuzz multiple targets per package (#46312)

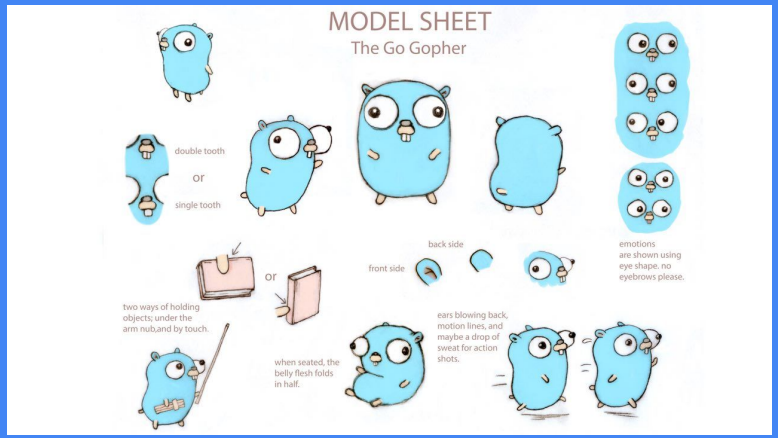
Suggestions

- *"**Fuzz targets should be fast and deterministic** so the fuzzing engine can work efficiently, and new failures and code coverage can be easily reproduced."*
- *"Since the fuzz target is invoked in parallel across multiple workers and in nondeterministic order, **the state of a fuzz target should not persist past the end of each call**, and the behavior of a fuzz target should **not depend on global state**."*

Demo



https://go.dev/blog/gopher



Useful links

- https://owasp.org/www-community/Fuzzing
- https://en.wikipedia.org/wiki/Fuzzing
- https://www.code-intelligence.com/blog/fuzzing-101-the-basics
- https://go.dev/doc/fuzz/
- https://go.dev/doc/tutorial/fuzz#installing_beta
- https://jayconrod.com/posts/123/internals-of-go-s-new-fuzzing-system
- https://go.googlesource.com/proposal/+/master/design/draft-fuzzing.md
- Official Go Fuzzing Slack channel: https://gophers.slack.com/archives/CH5KV1AKE

Q&A