WARSAW, JAN 15 2019

go-fuzz
or new unit testing

Oleg Kovalov

Allegro

Twitter:
oleg_kovalov

Github: cristaloleg

Me

- Gopher for ~3 years
- Open-source contributor
- Engineer at Allegro.pl core team

Twitter: @oleg_kovalov
Github: @cristaloleg

Everything start from the Wikipedia

Fuzzing is a software testing technique, often automated or semi-automated, that involves providing invalid, unexpected, or random data to the inputs of a computer program.

go-fuzz

- Made by The Dmitry Vyukov
  aka Bug Slaughterer at Google
- 300+ fixes in Go compiler and stdlib
- +inf in the wild, or more
- See AFL and syzkaller

What to test?

- text format/media codecs
- crypto
- network protocols
- compression
- compilers, interpreters, databases
- or anything where you can pass []byte

But why fuzzing?

- horribly easy to use
- no human interaction
- designed for computers

What it may (and will) find?

- out-of-bounds accesses
- nil derefs
- division by 0/floating-point
- infinite loops
- Segfaults (CGo)
- …

How does it work?

```
1.   Instrument program for code coverage
2.   Collect initial corpus of inputs

for {
  3. Randomly mutate an input from the corpus
  4. Execute and collect coverage

  if the input gives new coverage {
    5. Add the input to corpus
  }
}
```

One cozy loop

Bruteforce

```
func SafeFunc(input string) {
    if input[0] == 'A' {
        if input[1] == 'B' {
            if input[2] == 'C' {
                if input[3] == 'D' {
                    print(input[4])    // 😈
}}}}}

Brute force generation O(2^8^4) = O(2^32) tries.
```

“SafeFunc”

Smartforce

```
func SafeFunc(input string) {
    if input[0] == 'A' {
        if input[1] == 'B' {
            if input[2] == 'C' {
                if input[3] == 'D' {
                    print(input[4])    // 😈
}}}}}
```

Brute force generation O(2^8^4) = O(2^32) tries.

0. {}
1. {"A"}
2. {"A", "AB"}
3. {"A", "AB", "ABC"}
4. {"A", "AB", "ABC", "ABCD"}

Coverage-guided fuzzer needs O(4 * 2^8) = O(2^10) tries.

“SafeFunc”

So how to run it?

```
$ go get github.com/dvyukov/go-fuzz/go-fuzz
$ go get github.com/dvyukov/go-fuzz/go-fuzz-build

# build an executable
$ go-fuzz-build github.com/pkg/mypkg

# run fuzzing
$ go-fuzz -bin=./mypkg-fuzz.zip -workdir=workdir

# and follow the logs
workers: 8, corpus: 1525 (6s ago), crashers: 6, execs:
0 (0/sec), cover: 1651, uptime: 6s

workers: 8, corpus: 1525 (9s ago), crashers: 6, execs:
16787 (1860/sec), cover: 1651, uptime: 9s

workers: 8, corpus: 1525 (12s ago), crashers: 6,
execs: 29840 (2482/sec), cover: 1651, uptime: 12s
```

Fuzzing

func Fuzz([]byte) int

```
// +build gofuzz

package mypkg

func Fuzz(data []byte) int {

    _, err := WellTestedFunc(string(data))
    if err != nil {
      return 0
    }
    return 1
}
```

95% fuzz funcs

Best practices

- do not run on each build
- but run regularly
- fuzz 1 func at time
- it’s not unit test replacement
- SecOps be aware 😏

(doesn’t work with go modules?)

That’s all folks

Thank you
Questions?

Twitter: @oleg_kovalov
Github: @cristaloleg