Lecture 8

maandag 5 december 2022 17:32

Stroustrup (C++): no additional complexity should be incurred (performance penalty)

Gosling (Java): avoid common problems by making language safe

fuzzing : execution of PUT using randomly sampled inputs

fuzz testing : use of fuzzing to test

fuzzer : program performing fuzz testing

fuzz campaign : specific execution with security policy

super important → bug oracle : determines policy violation

fuzz configuration : parameter values controlling algorithm

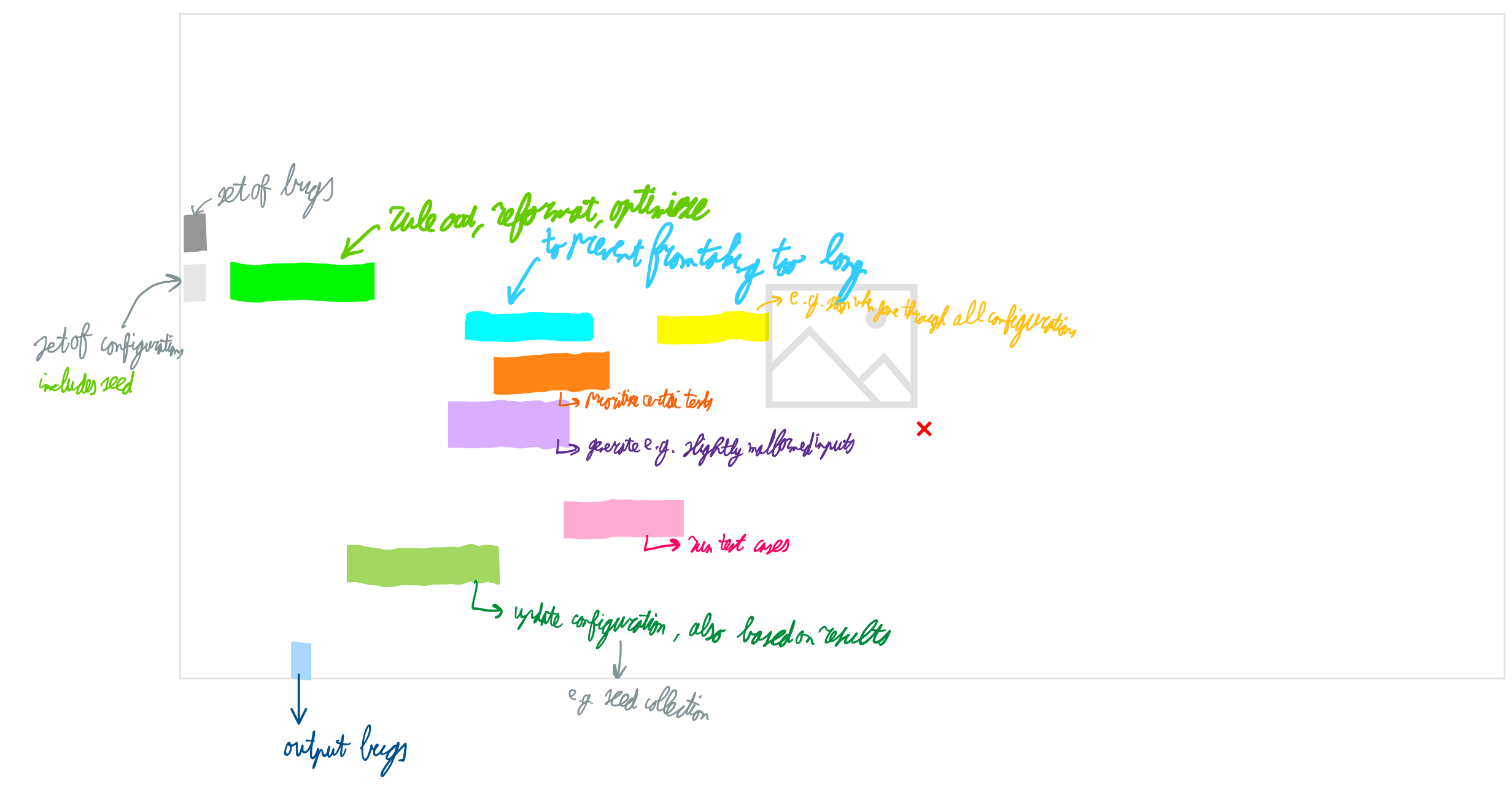
super important → seed : input to PUT, used for generating test cases by modifying seed

seed pool : collection of seeds, which can evolve throughout fuzz campaign

most of input is correct → brings program into specific state

modify small part → introduce error

there's a trade-off between more specific input and likelihood of finding bugs



black box : no investigation of PUT / do not know internals of PUT also: model for attacker

white box : can explore PUT state space

can instrument PUT for additional info

more powerful; not always clear what to do with this power

note: can include both reading & writing PUT

Grey-box: middle ground

e.g. might know about code coverage

or: third-party libraries don't have source code available

Pre-process : input = configs
output = new set of configs

can also involve decreasing seed pool size to optimize   for example: remove seeds that needlessly go through 'loops' in state machine

minset : minimal set of seeds that maximizes coverage metric

or: build wrapper around library

prepare model for future input generation

schedule : address fuzz configuration scheduling (FCS) problem

→ pick configuration which likely leads to most favorable outcome

note: time can be used

- to gather more information, to improve future decision   explore
- fuzzing configurations and find favorable outcomes   exploit

suggestion:

- check configuration with control-flow jumps

inputgen : turn configuration into executable test cases

need to take into account that program might exclude faulty input

suggestion: create faulty encoder

mutation-/seed-based fuzzers: slightly modify correct input to make it faulty

bit flip
arithmetic mutation
⋮

many techniques

- sometimes: modify PUT

inputeval: execute PUT; use bug oracle

- → crash
- → memory error
- → undefined behavior
- → semantic difference: check whether other program does the same

use forking to optimize execution, or restore memory image

triage: analyzing + reporting problems

fuzzer taming problem: prioritization / ranking of bugs (uniqueness + severity) → exploitability

deduplicate bugs

find part of input that causes bugs

confupdate : update set of configurations

black-box: does not change much

grey-/white-box: more sophisticated

maintaining minset → minimal set of test cases which maximizes coverage

evolutionary algorithm : mutation, recombination, selection