code intelligence

Introduction to modern fuzzing

Find and fix vulnerabilities before they reach production

Jochen Hilgers
Senior Software Developer

Background

- Master Computer Science
- Backend- / Web-Development
- Focus on Software Quality

Responsibilities at Code Intelligence

- CLI Tools

✉ hello@jochen.dev jochil

Today’s Topics

1. What is Fuzzing
    a. History
    b. Classification
2. Modern Fuzzing
    a. Where to use it
    b. What kind of bugs / vulnerabilities can it find
3. Live Demos

What’s all the fuzz about

code intelligence

1 800
MICROSOFT OFFICE

12 000
LINUX

25 000
GOOGLE CHROME

37 000
OSS-FUZZ

5 200
MOZILLA FIREFOX

Finding Heartbleed

This tutorial will show you how to find Heartbleed using libFuzzer and ClusterFuzz.

ShellShock / Bashdoor

50 CVEs in 50 Days: Fuzzing Adobe Reader

December 12, 2018

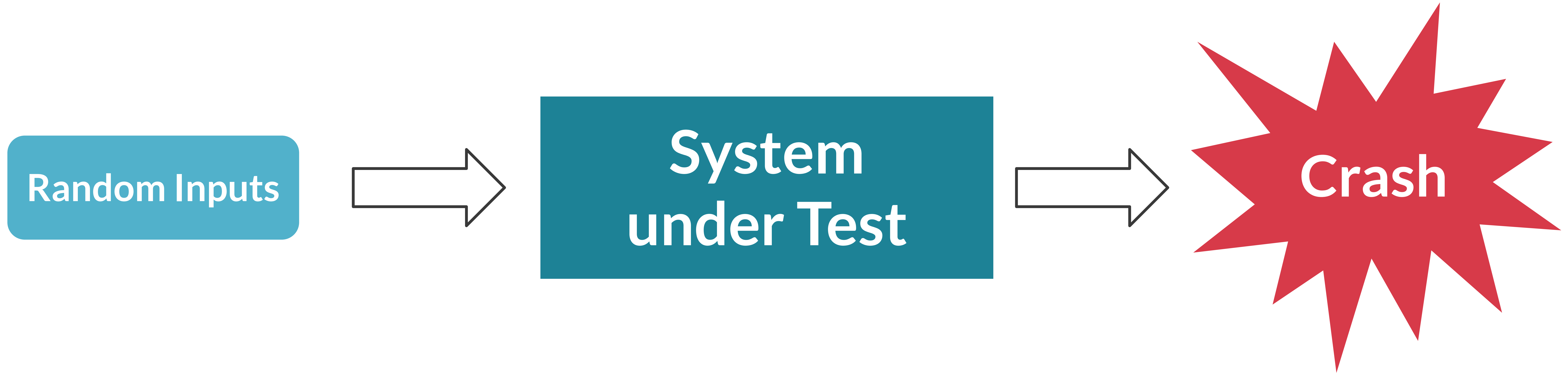
Research By: Yoav Alon, Netanel Ben-Simon

What is Fuzzing

fuzz | verb.

/ˈfəz/

1. to make or become blurred



The History of Fuzzing

code intelligence

- Random Testing is around since the 1950s
- Fuzz Testing (Fuzzing) originated around 1988
  - https://pages.cs.wisc.edu/~bart/fuzz/ (project still active)
- 2012 Google announces ClusterFuzz
- 2013 first release of American fuzzy lop (AFL)
  - 2014 Shellshock most vulnerabilities discovered by AFL
- 2016 libFuzzer part of LLVM/clang
- 2016 Google announces OSS-Fuzz
- 2020 Microsoft releases OneFuzz

Classification

… by Target

- Application fuzzing
- Protocol fuzzing
- File format fuzzing

… by knowledge of program structure (*-box)

- white
- grey
- black

… by how input is generated

- Generation-based fuzzing
  Generating input based on a model or grammar (eg. source code, file formats, …)
- Mutation-based fuzzing
  Mutating input (bit flipping, …)

… by input structure awareness

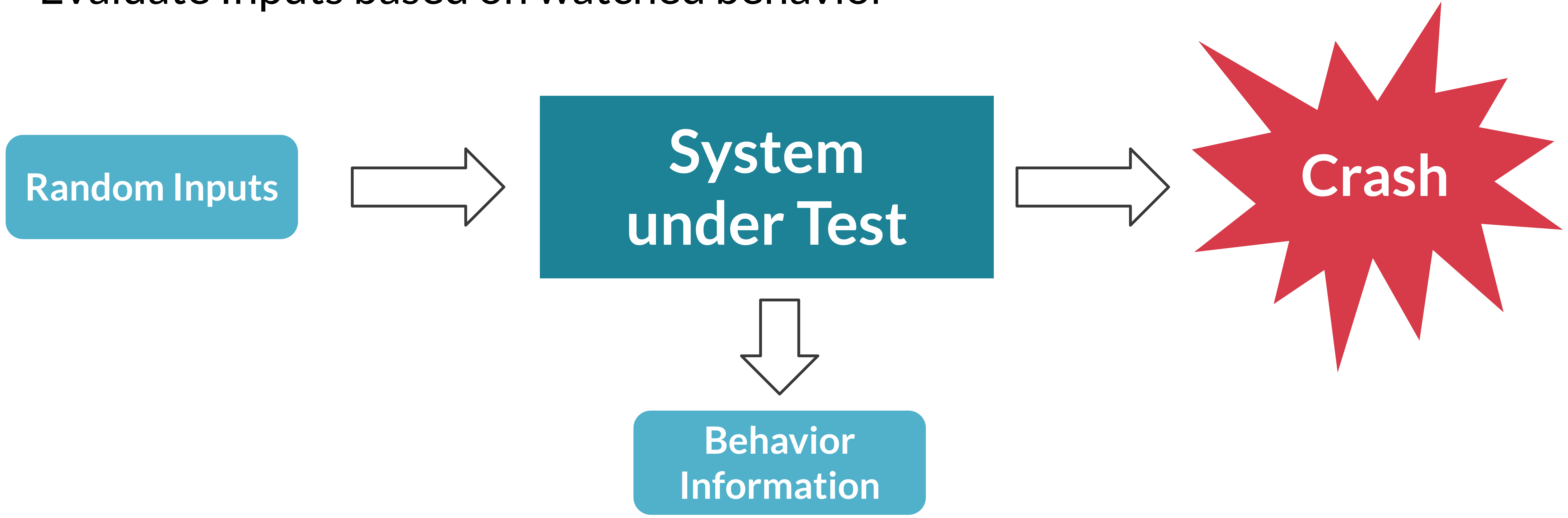
- Smart fuzzing
- Dump fuzzing

The next step to modern fuzzing

code intelligence

(Coverage) Guided mutation-based fuzzing

- Generating new input from existing one (bit flipping, evolutionary /genetic algorithms, …)
- Evaluate Inputs based on watched behavior



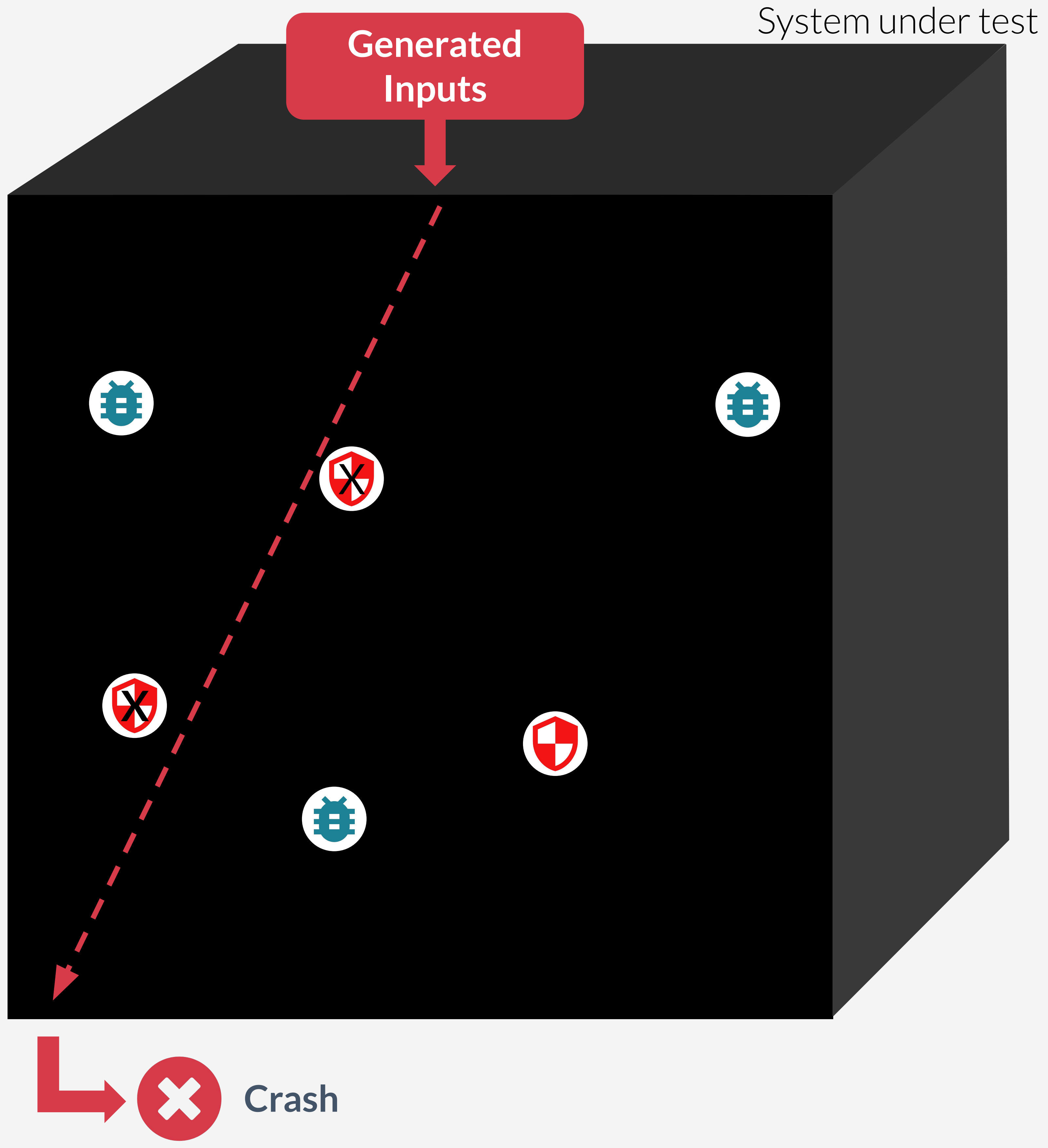
Black-box vs Coverage-guided fuzzing

code intelligence

Bugs   Vulnerabilities

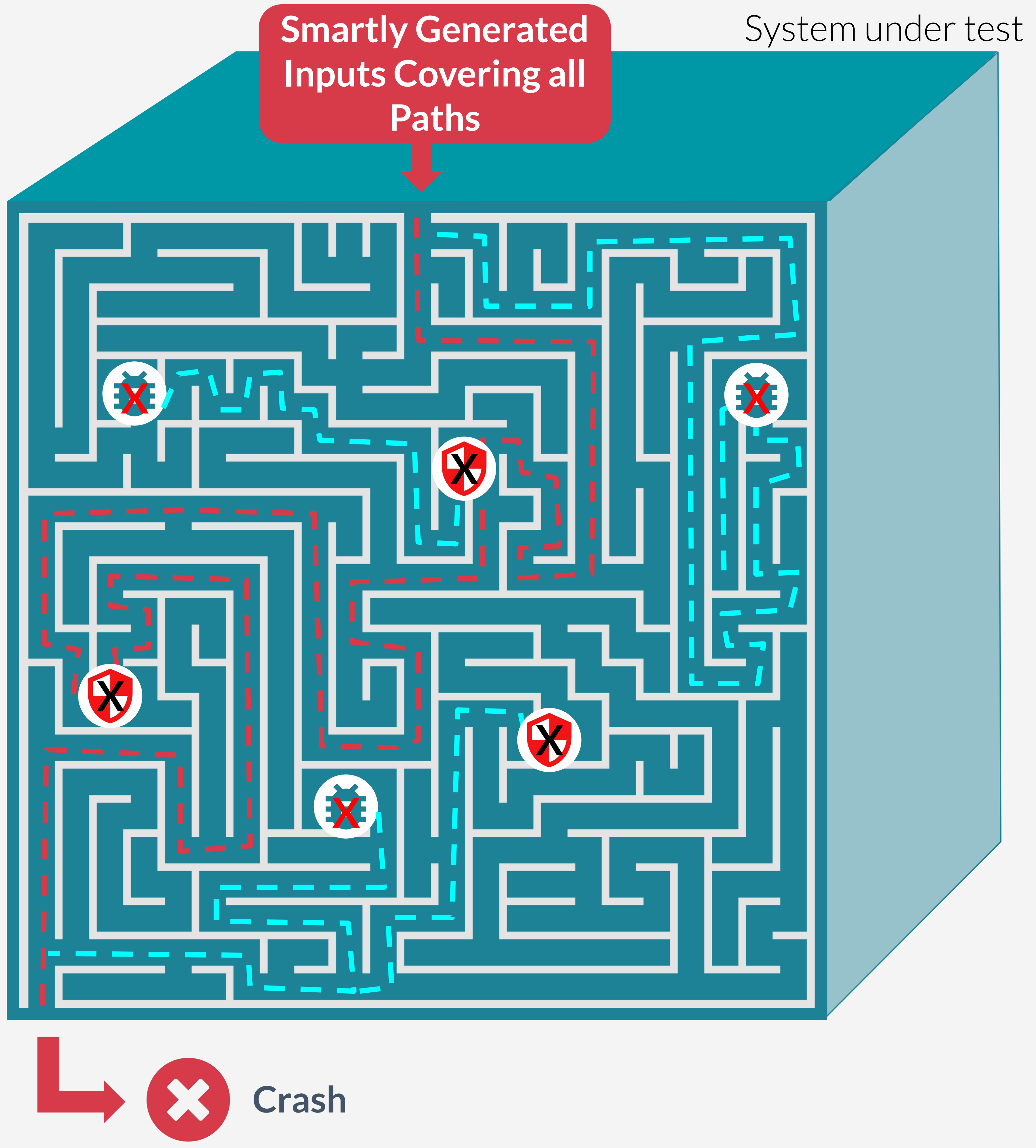
Black-box Fuzzing

- No knowledge of which code is reached
- Misses critical bugs



Coverage-guided Fuzzing

- Intelligent & feedback-driven mutations
- Maximizes code coverage



In- vs. Out-Of-Process Fuzzing

code intelligence

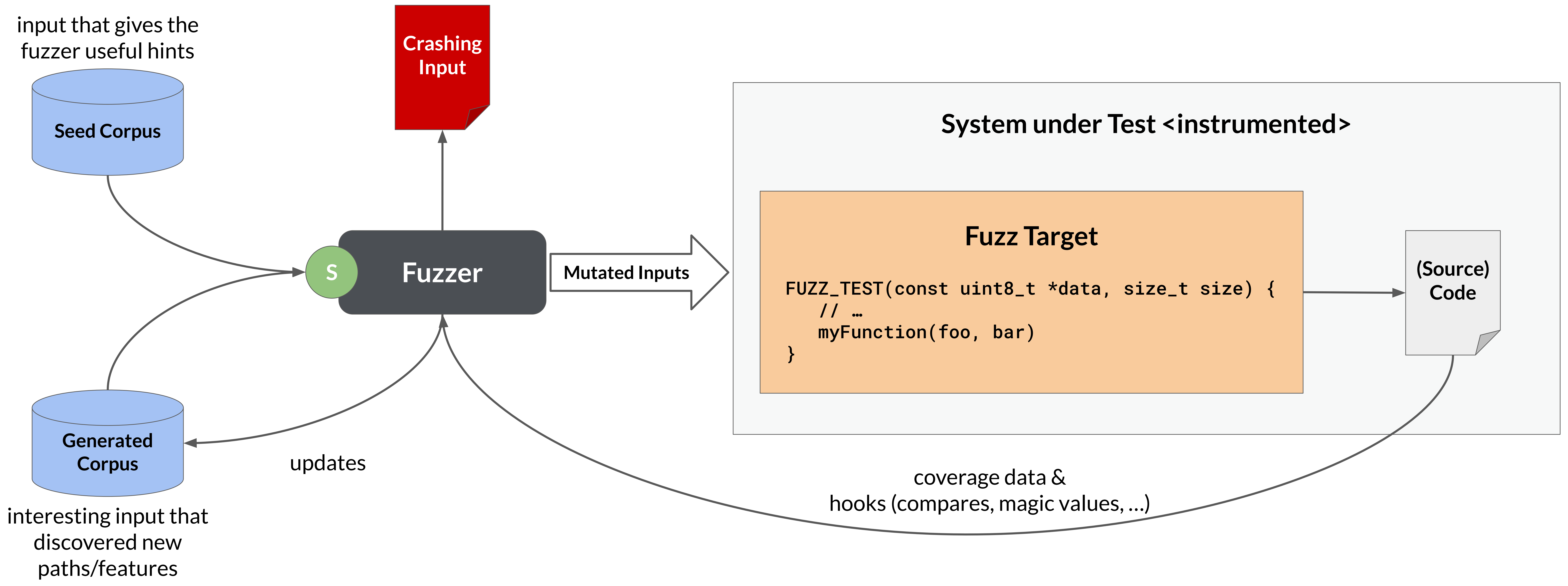
In-process fuzzing

- Fuzzer runs in the process (or VM) of the SUT
- Fast
- Relatively easy to get information

Out-of-process fuzzing

- Fuzzer runs alongside the SUT
- Usually a little bit more "communication" overhead
- Useful for distributed systems, if used with "talk back" channel
    - Microservices
- Often used for protocol fuzzing

Components of a Fuzzing System


input that gives the fuzzer useful hints
Seed Corpus
Crashing Input
S
Fuzzer
Mutated Inputs
System under Test <instrumented>
Fuzz Target
(Source) Code
Generated Corpus
updates
coverage data & hooks (compares, magic values, …)
interesting input that discovered new paths/features

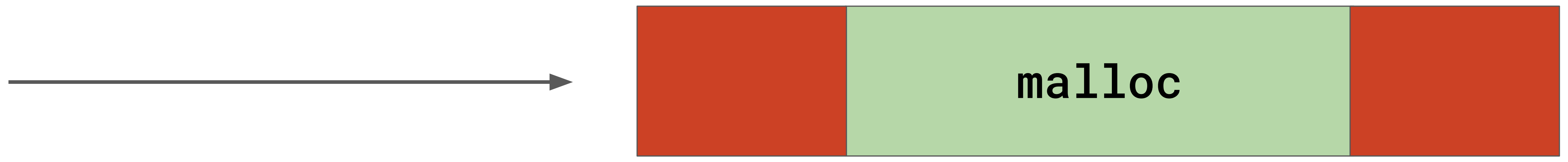
```
FUZZ_TEST(const uint8_t *data, size_t size) {
   // …
   myFunction(foo, bar)
}
```

Sanitizers

Needed for actual triggering the Bug / Vulnerability

Common (C/C++) Sanitizer:

- AdressSanitizer (ASan)
- MemorySanitizer (MSan)
- UndefinedBehaviorSanitizer (UBSan)
- ThreadSanitizer (TSan)



What type of bugs / vulnerabilities ...

- ASan
  - (Global|Heap|Stack) Buffer Overflow
  - Use after (return|free|scope)
  - Memory Leaks
  - ...
- MSan: reads of uninitialized memory
- TSan: Data Races, Deadlocks
- UBSan
  - Signed integer overflow
  - Out of bounds (Array/BitShifts)
  - Floating point conversion overflow
  - Dereferencing misaligned or null pointers
  - ...
- Resource usage bugs: Memory exhaustion, hangs or infinite loops, infinite recursion
- Logical bugs:
  - Discrepancies between two implementations of the same protocol
  - Round-trip consistency bugs (e.g. compress the input, decompress back, - compare with the original)
  - ...
- Plain Crashes: NULL dereferences, Uncaught exceptions

Good Targets for Fuzzing

Anything that consumes untrusted or complicated inputs!

- Parsers
- Media codecs
- Network protocols, RPC libraries
- Crypto
- Compression
- Compilers and interpreters
- Regular expression matchers
- Text/UTF processing
- Databases
- Browsers
- Text editors/processors
- OS Kernels, drivers, supervisors and VMs

Limitations

- Non-crashing bugs are hard to find
- Fuzzing without Sources makes everything harder
    - Coverage Informations
    - Input formats
    - “Brandon Falk - Adventures in Fuzzing” https://www.youtube.com/watch?v=SngK4W4tVc0

Demo Time

code intelligence

Jazzer - Modern Fuzzing for JVM

code intelligence

- Coverage-guided: based on libFuzzer & JaCoCo
- In-process: very fast (up to 1M executions/s)
- Open source since Feb 2021
- Powers JVM Fuzzing in Google's OSS-Fuzz
- Autofuzz mode
- Hooks for own Sanitizer/Bug detectors

github.com/CodeIntelligenceTesting/jazzer

Why fuzz memory-safe languages

code intelligence

Functional Bugs

- Uncaught exceptions
- Assertions
- Inconsistent implementations (differential fuzzing)
- Property-based testing

Security Issues

- Infinite loops
- OutOfMemoryError
- Remote Code Execution
- Path Traversal
- Injections into Domain Specific Languages (SQL, EL, Scripts, …)
- ...

Demo Jazzer

github.com/CodeIntelligenceTesting/jazzer

cifuzz - Fuzzing as easy as writing unit tests

- Goals
  - Writing Fuzz tests should as easy as writing unit tests
  - One convenient CLI tool, no matter if you are working with C++ or JavaScript
- Under active development
- Open source from the start

- Features by now
  - C/C++ including very comfortable CMake integration
  - IDE Integration (CLion, vscode)
  - Coverage reporting
  - Sandboxing (linux)
  - Regression testing
  - Findings management
- Soon
  - Java / Jazzer Support
  - JavaScript / Jazzer.js Support
  - Out-of-the-box Debugging
  - SaaS Connection

github.com/CodeIntelligenceTesting/cifuzz

Demo cifuzz

github.com/CodeIntelligenceTesting/cifuzz

Jazzer.js

- Coverage-guided, in-process fuzzer for node.js
- We will releasing it next week as open source

github.com/CodeIntelligenceTesting/jazzer.js

Ganz neu - Jazzer.js

github.com/CodeIntelligenceTesting/jazzer.js

Some tips for the start

- Try to optimize coverage
  - give hints to the fuzzer (seed corpus, internal structures)
- It helps a lot if your fuzz test/target is... (same for the SUT ;))
  - deterministic (eg. not changing global state)
  - stable and fast
  - not wasting resources
- Use FuzzedDataProvider, especially for C/C++ :)

Some Sources

- https://github.com/secfigo/Awesome-Fuzzing
- https://github.com/google/fuzzing
- https://www.code-intelligence.com/blog

We are hiring ... of course ;)

code intelligence

- Senior Go Developer (d/f/m)
- Senior Fuzzing Expert (d/f/m)
- (Senior) Clojure Developer (d/f/m)
- Senior Backend Developer (d/f/m)
- Application Security Engineer / Pentester / DevSecOps (d/f/m)
- ... and a few more

https://www.code-intelligence.com/careers

code intelligence

Thanks for joining me!

Any Questions?