

Prerequisites

- Linux, macOS (Intel & M1), Windows
- Docker
- IDE with JUnit 5 support
 - IntelliJ
 - VS Code with the "Test Runner for Java" extension
 - Eclipse (supported, but not covered by examples)
- Clone github.com/CodeIntelligenceTesting/jazzer-workshop
- A favourite Java library, ideally one that handles untrusted input

Slides available at:

kl.rs/jug.ch





code intelligence

Fuzzing Java with Jazzer



Fabian Meumertzheim

Senior Software Engineer



Background

- Mathematician by education
- OSS contributor (Bazel, Chromium, Android Password Store)

Responsibilities at Code Intelligence

- Fuzzing Technologies
- OSS Initiatives & Cooperations

 @fhenneke

 meumertzheim@code-intelligence.com

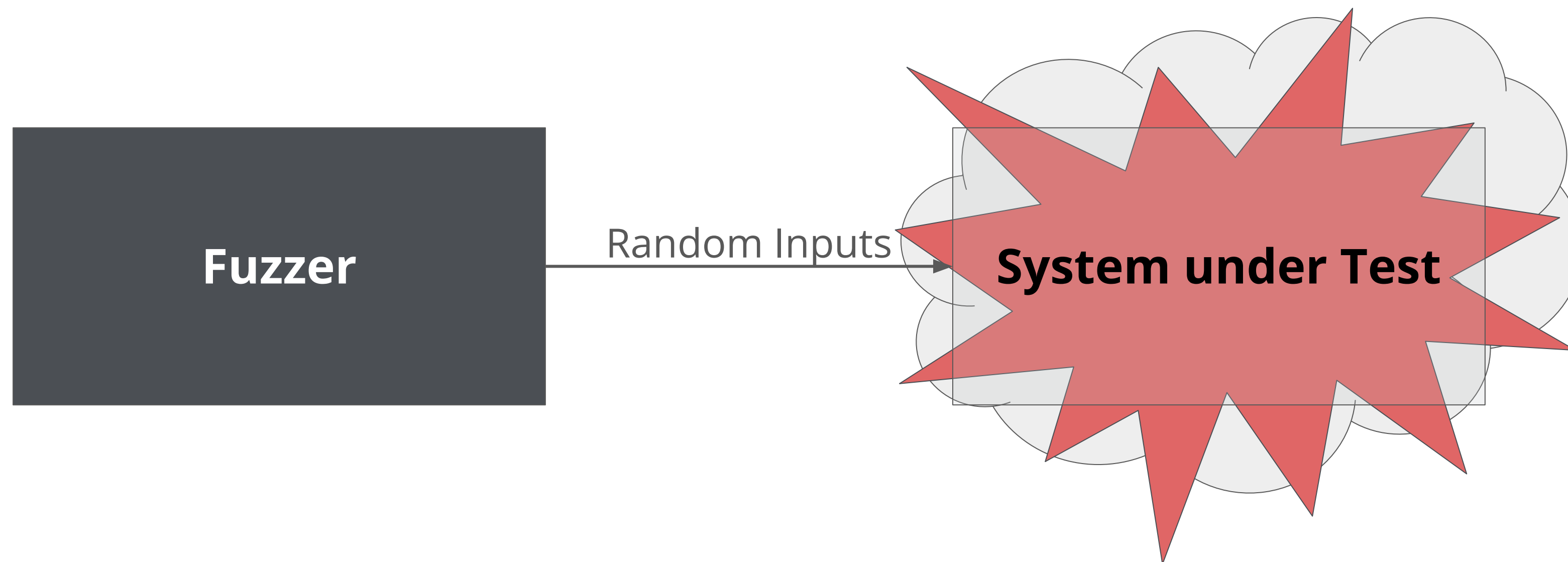
 fmeum

What is Fuzzing?

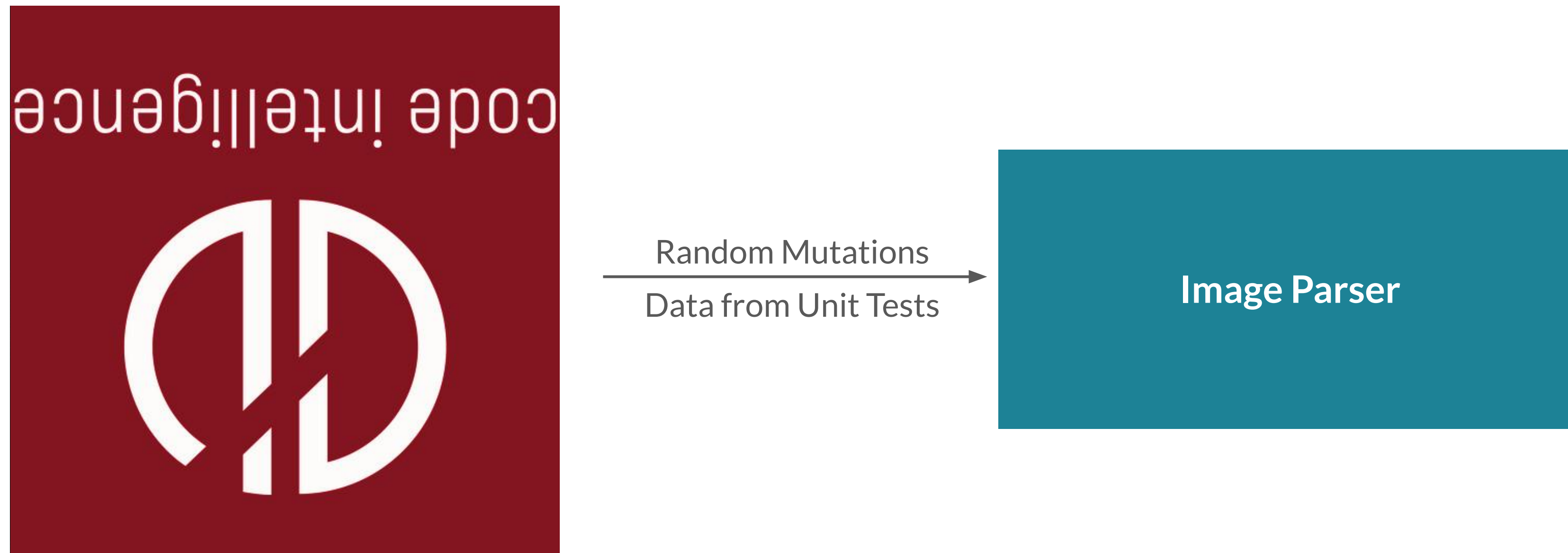
fuzz | verb.

/'fəz/

1. to make or become blurred



Blackbox Fuzzing



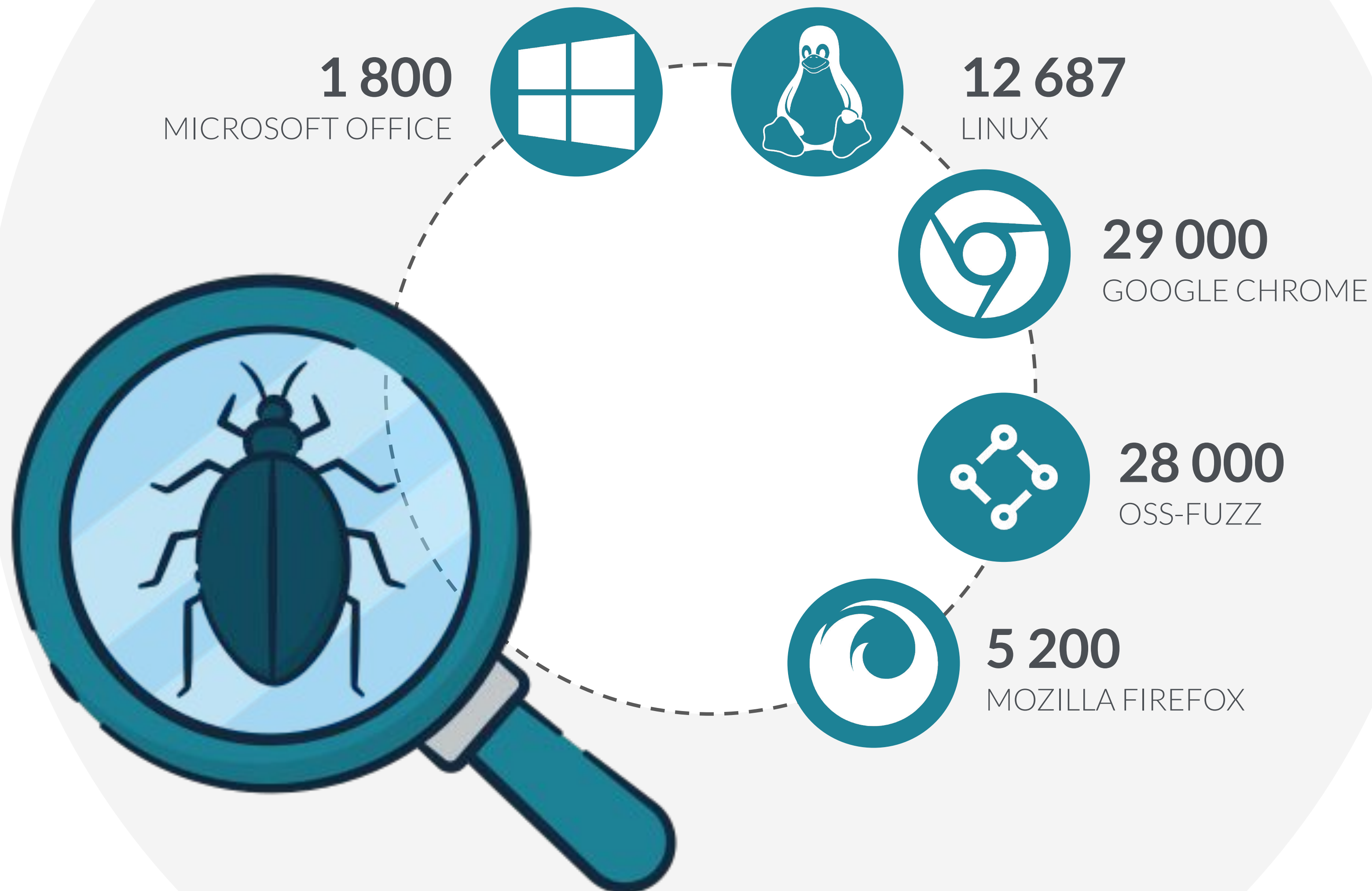
Whitebox Fuzzing



covered branches, magic bytes,
compared values

```
private static final int[] MAGIC_NUMBERS_GIF = { 0x47, 0x49, };  
private static final int[] MAGIC_NUMBERS_JPEG = { 0xff, 0xd8, };  
  
//...  
if (compareBytePair(MAGIC_NUMBERS_GIF, input)) {  
    return ImageFormats.GIF;  
} else if (compareBytePair(MAGIC_NUMBERS_JPEG, bytePair)) {  
    return ImageFormats.JPEG;  
}  
return ImageFormats.UNKNOWN;  
//...
```


What's all the Fuzz About?



Finding Heartbleed

This tutorial will show you how to find [Heartbleed](#) using libFuzzer and ClusterFuzz.

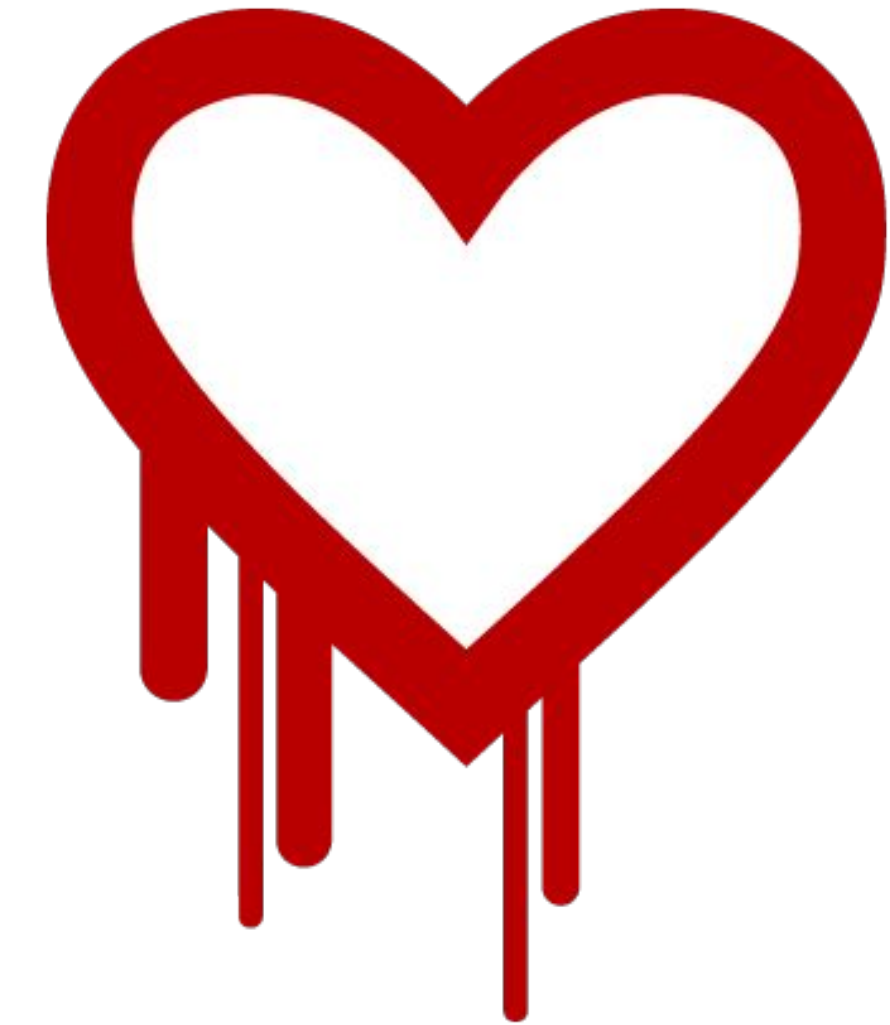


50 CVEs in 50 Days: Fuzzing Adobe Reader

December 12, 2018

Research By: Yoav Alon, Netanel Ben-Simon

Heartbleed (CVE-2014-0160)



- Buffer Over-Read in the TLS implementation of OpenSSL $\leq 1.0.1f$
- Allows partial disclosure of memory contents (e.g. private keys)
- Introduced: 2011-12-31
- Reported: 2014-04-01
- ~17% of the public servers using TLS were affected
- Lessons learned:
 - OSS development of core infrastructure is severely underfunded (2 full-time devs for 500k SLOC)
 - Code reviews are not sufficient by themselves to catch security issues during development

Responsible Disclosure for OSS software

If you have found an issue with security impact in an open-source project:

1. Keep it to yourself.
2. Send detailed instructions reproducing the issue to (ordered by preference):
 - a. Dedicated security contacts (look for SECURITY.md, [security.txt](#), [Tidelift](#), bug bounty programs)
 - b. Maintainers (README.md, backing company website, releases)
 - c. Top contributors (commit history)
 - d. Corporate sponsors
3. Wait. Most open-source projects are freetime projects, even if they are critical infrastructure.
4. Follow up after a few weeks and go back to 2.
5. Ask your contact whether they will [request a CVE](#) or would prefer you to do so. CVEs aren't badges, but greatly simplify the "Am I affected?" problem for end users.
6. Agree on a date for public disclosure.

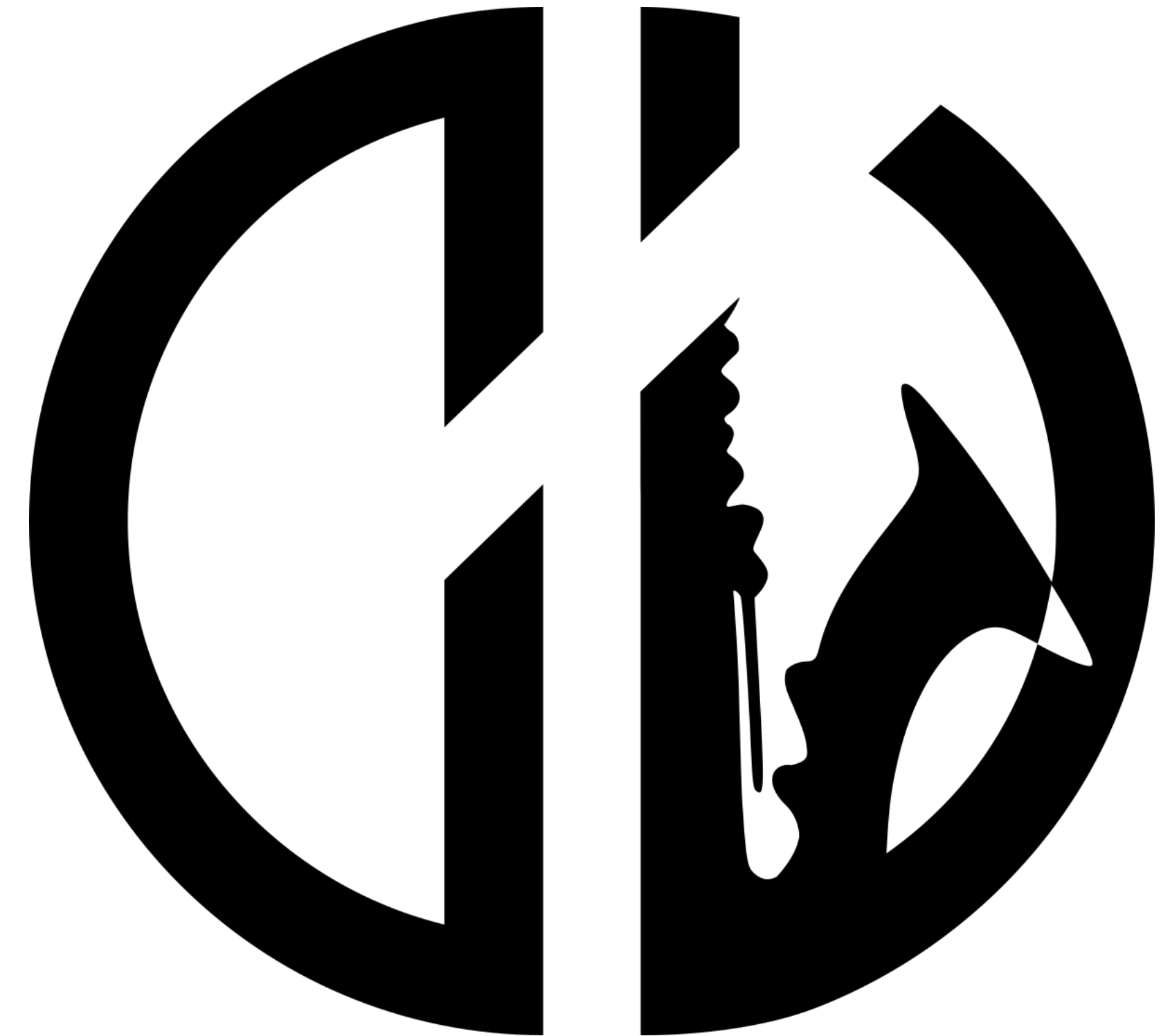
Getting to Know Jazzer

Jazzer — Modern Fuzzing for the JVM

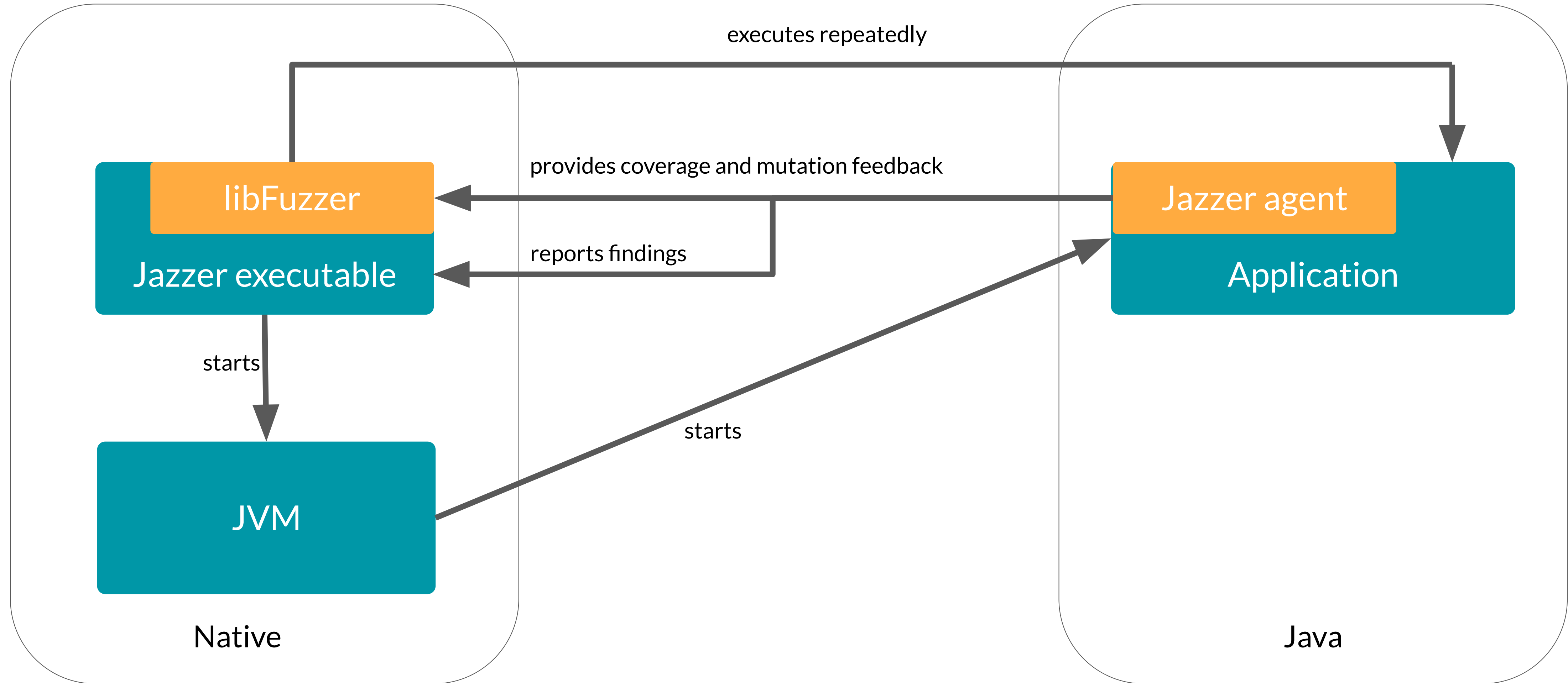
- Coverage-guided: based on libFuzzer & JaCoCo
- No sources required: agent-based instrumentation
- Collects dynamic data from comparisons & common functions
- Open-source since Feb 2021



github.com/CodeIntelligenceTesting/jazzer



What's in a Jazzer?



Autofuzz – “No-Code Fuzzing”

Autofuzz automatically generates arguments for a specified method, reporting all uncaught exceptions and bug detector findings.

Autofuzz generates:

- primitive types
- Strings
- Arrays and maps
- InputStreams
- Class and Method instances
- User-defined classes via recursive constructor and builder invocations

Getting Started with Autofuzz

```
docker run \  
  -v $(pwd):/fuzzing \  
  -it cifuzz/jazzer-autofuzz \  
  org.jsoup:jsoup:1.14.1 \  
  "org.jsoup.Jsoup::parse(java.lang.String)"
```



[https://jsoup.org/apidocs/org/jsoup/Jsoup.html#parse\(java.lang.String\)](https://jsoup.org/apidocs/org/jsoup/Jsoup.html#parse(java.lang.String))

Optional arguments:

```
--keep_going=N           # Stop after N findings  
--autofuzz_ignore=some.Exception,some.other.Exception
```


Getting Started with Autofuzz

Task:

- Try out Autofuzz on your favourite Java library.
 - Functions that "parse" their arguments make for good entry points.
 - The smaller the scope, the better.
- If there are findings, compile and run the Java reproducer (Crash_<hash>.java)
 - Requires having the library and its dependencies on the classpath.

Questions:

- Does the fuzzer produce any interesting finding?
- Was Autofuzz able to construct reasonable arguments?

Writing a First Java Fuzz Test

Fuzzing Terminology

- **fuzz test (also fuzz target)**
 - function or program that receives input from the fuzzer
 - should exercise interesting and varied behavior in the code under test
- **crashing input**
 - fuzz target input that causes a crash (or more generally a bug) in the tested code
- **seed corpus**
 - user-supplied collection of valid inputs (e.g. example JPEGs for an image parser)
 - base for the fuzzer's mutations
- **generated corpus**
 - collection of "interesting" inputs generated by the fuzzer over time
- **sanitizer (also bug detector)**
 - instrumentation applied to the code under test that detects undesired behavior as it happens
 - first examples: AddressSanitizer, UndefinedBehaviorSanitizer for C/C++

Creating a Fuzz Test

1. Set up JUnit 5 for your project.
 - Many IDEs such as IntelliJ and VS Code handle this automatically.
 - For others, start with one of the samples at github.com/junit-team/junit5-samples.
2. Add a dependency on `com.code-intelligence:jazz-junit:0.12.0`.
3. Add a test method annotated with `@FuzzTest` taking a single parameter of type `byte[]` or `FuzzedDataProvider` (more on that soon).

```
15 package com.example;
16
17 import com.code_intelligence.jazzer.api.FuzzedDataProvider;
18 import com.code_intelligence.jazzer.junit.FuzzTest;
19
20 class ExampleFuzzTests {
21     @FuzzTest
22     void dataFuzz(FuzzedDataProvider data) {
23     }
24 }
```

FuzzedDataProvider

Problem:

Fuzzers naturally produce **raw bytes** as input, but Java functions rarely operate on `byte []`.

Solution:

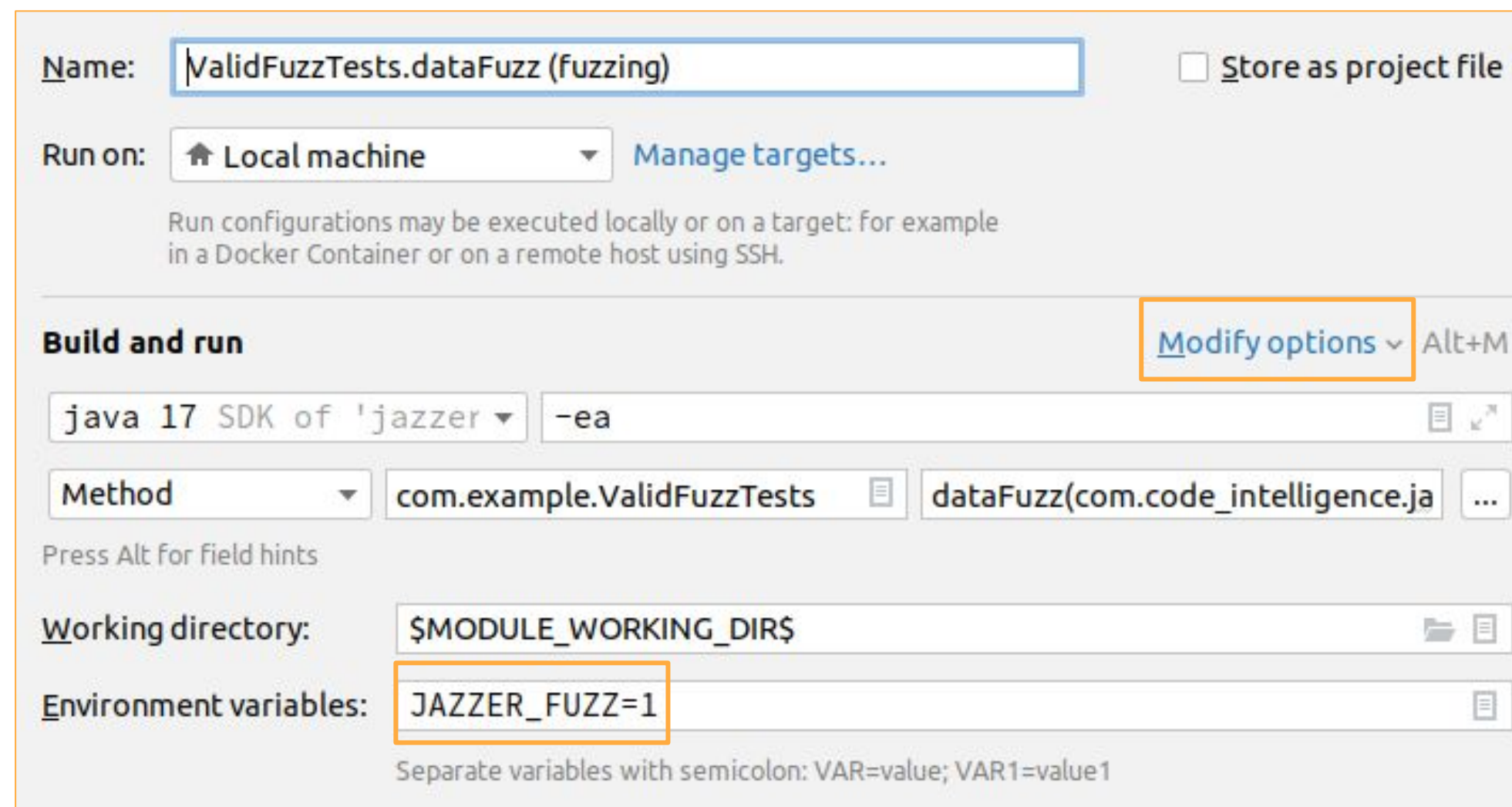
Jazzer offers a `FuzzedDataProvider` that turns the binary input into Java types.

See the [javadocs](#) for a full reference of all functions.

<code>java.lang.String</code>	<code>consumeRemainingAsString()</code>
<code>short</code>	<code>consumeShort()</code>
<code>short</code>	<code>consumeShort(short min, short max)</code>
<code>short[]</code>	<code>consumeShorts(int maxLength)</code>
<code>java.lang.String</code>	<code>consumeString(int maxLength)</code>
default boolean	<code>pickValue(boolean[] array)</code>
default byte	<code>pickValue(byte[] array)</code>

Running a Fuzz Test from the IDE

- Just click the play button to execute the test on the *seed corpus* (currently empty) only.
 - This mode serves as a regression test that behaves just like an ordinary JUnit test.
- To start fuzzing, set the environment variable `JAZZER_FUZZ` to any non-empty value.
 - IntelliJ: Edit Configurations → Modify options... → Environment variables
 - VS Code: Add a `java.test.config` to your `settings.json`



```
60     ... "java.test.config": [  
61     ...     {  
62     ...         "name": "default",  
63     ...         "workingDirectory": "${workspaceFolder}"  
64     ...     },  
65     ...     {  
66     ...         "name": "fuzzing",  
67     ...         "workingDirectory": "${workspaceFolder}",  
68     ...         "env": {"JAZZER_FUZZ": "true"},  
69     ...     }  
70     ... ],
```


The Two Faces of @FuzzTest

Regression test (JAZZER_FUZZ not set)

- an ordinary JUnit `@ParameterizedTest`
- executes the fuzz test in a subtest for each entry in the seed corpus directory (`YourClassNameSeedCorpus` in your fuzz test's package's resource directory, change with `seedCorpus`)
- meant to be run in CI alongside your usual tests to verify that all issues found with fuzzing have been and remain fixed
- can also be used to conveniently debug findings and compute coverage

Fuzzing run (JAZZER_FUZZ non-empty)

- performs actual fuzzing
- starts by running over the seed corpus
- passes if it doesn't find a crash within 1 min (default, use `maxDuration` to change)
- emits crashing inputs into the seed corpus directory to have them picked up by the regression test
- can be run in CI, but mostly meant for local usage

Create and Run Your First Fuzz Test

Tasks:

- Start with the Maven project template in the “fuzz-test-template” directory.
- Execute the fuzz test from the IDE, both as a regression test (without JAZZER_FUZZ) and as an actual fuzzing run (with JAZZER_FUZZ).
- Debug a crashing input by debugging the corresponding failing subtest.
- Add some code under `src/main/java/com/example` and another `@FuzzTest` that tests it.
- Use more of the methods provided by `FuzzedDataProvider`.

Running a Fuzz Test from the Terminal

1. Download a Jazzer release or use the cifuzz/jazzer image (working directory: / fuzzing)
2. Run the `jazzer (.exe)` executable with the following arguments:
 - `--cp=<classpath>`: classpath including your tests and tested code (no need to list `jazzer-junit`)
 - `mvn test -X` prints the classpath under "test classpath"
 - `--target_class=<name>`: name of the class containing the `@FuzzTest`
 - `--target_method=<name>`: name of the particular `@FuzzTest` to run in that class (if multiple)
 - `<path to generated corpus>`: directory to collect fuzzer-generated inputs in

Some features as well as optimal performance are (currently) only available in this way:

- `--fork=N`: fuzz in N parallel processes
- `--minimize_crash=1 <input file path>`: minimize a given crashing input

Running a Fuzz Test from the Terminal

Tasks:

- Build your fuzz test and collect the required classpath.
- Execute the fuzz test using the Jazzer CLI.
- Try out `--fork`.
- Try to minimize a finding with `--minimize_crash`.

Advanced Fuzzing Techniques

Military-Grade Encryption™

Task:

- Execute the fuzz test in the "encryption" project.

Questions:

- How far does the fuzzer get?
- Why does it get stuck?
- What would it need to know to progress?

Through the Maze

Task:

- Execute the fuzz test in the "maze" project.

Questions:

- How far does the fuzzer get?
- Why does it get stuck?
- What would it need to know to progress?

Writing Custom Bug Detectors

Concept:

- Fuzzing is already good at dynamically exploring code and program state.
- Thus, all it takes to find vulnerabilities is to *detect them when they happen*.

Current state:

Jazzer ships with bug detectors for:

- Insecure deserialization
- Insecure use of reflection (arbitrary class loads/method executions)
- OS command, SQL, LDAP and Expression Language injection
- NamingContextLookup (of log4shell fame)

Writing bug detectors isn't that difficult – we will go over an example.

“Bring Your Own Library”

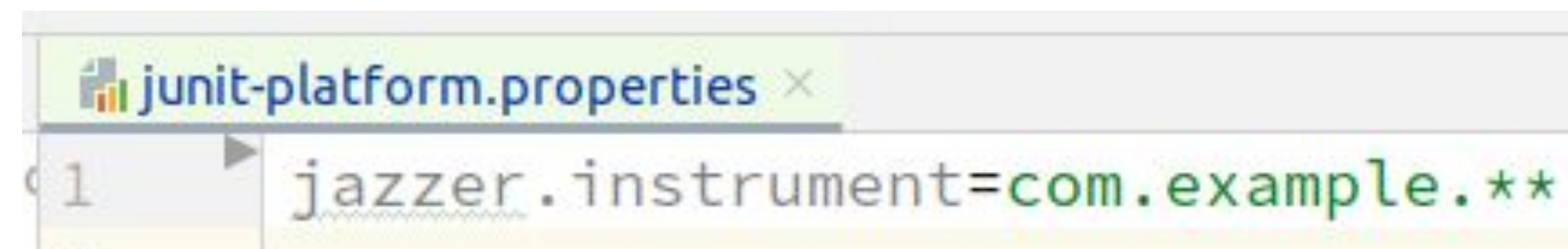
Fuzzing an OSS Library

Tasks:

- You can again start with the Maven project template in the “fuzz-test-template” directory.
- Add a dependency on your favourite Java library and write a `@FuzzTest` covering an “interesting” function: a parser, an algorithm, a sanitization/canonicalization routine, ...
- Verify that the result is as expected and let the fuzzer try to break these assumptions.

Note:

By default, Jazzer only instruments code in packages that share the first two segments (e.g. `com.example`) with your test. If needed, modify the default by specifying a comma-separated list of package globs in `junit-platform.properties`:



```
junit-platform.properties
jazzer.instrument=com.example.**
```

Adding a Seed Corpus

Problem:

When fuzzing complex formats, synthesizing valid and varied inputs from scratch is difficult.

Solutions:

Add example inputs to the *seed corpus*.

- JUnit: Create and populate the test resource directory
`src/test/resources/com/example/YourFuzzTestClassSeedCorpus`
- CLI: Pass path to seed corpus to Jazzer as the second positional argument after the generated corpus.

Good starting points for common formats:

- github.com/dvyukov/go-fuzz-corpus
- github.com/MozillaSecurity/fuzzdata/tree/master/samples

Task: Add seed corpus entries to your fuzz test and observe the difference in coverage.

Where to go from here

OSS-Fuzz – Large-Scale (Java) Fuzzing

OSS-Fuzz is Google’s fuzzing initiative for open-source software (est. 2016).

Jazzer has been available on OSS-Fuzz since March 2021.

Current stats:

- **66** Java projects
 - including: jsoup, Jackson, zxing, protobuf-java, ...
- **>500** bugs found
- **>100** issues with security impact
- **17** CVEs for released security-critical bugs
- Happy maintainers due to highly actionable bug reports with automatic fix verification



OSS-Fuzz – How to Contribute

Integrating a relevant open-source project pays **\$5,000 or more** and minimally comes down to:

1. Contact maintainers to gauge interest up front.
2. Provide a **Dockerfile** describing the project's build-time dependencies (usually just Maven/Gradle).
3. Add **fuzz targets** covering the basic use cases of the project that operate on **untrusted data**.
4. Add a **build.sh** that builds the project and fuzz targets from source.
5. (Optional) Upstream the fuzz targets.
6. (Recommended) Donate to the project — or be a maintainer and use this as additional funding.

More info at: google.github.io/oss-fuzz/getting-started/new-project-guide/jvm-lang

cifuzz – Open-Source CLI for Fuzzing

- A single CLI tool for the entire fuzzing workflow:
 - Creating fuzz test stubs & setting up IDE/build system integrations
 - Running fuzz test and managing findings
 - Generating coverage reports
- Currently supports CMake (C/C++), next up on the roadmap:
 - Bazel (C/C++, Java)
 - Maven/Gradle
 - Node.js (backed by [Jazzer.js](#))

github.com/CodeIntelligenceTesting/cifuzz

CI Fuzz – Web Apps, CI/CD & more

- Fuzzes multi-service, multi-language deployments (REST/gRPC, C++/Java/Go)
- Integrates with your CI/CD pipeline

PROJECT
WebGoat

EXPORT PDF | master with 3 findings

Type	Severity ↑	Location	
SQL Injection	Critical	webgoat-lessons/.../SqlInjectionLesson4.java:52	DEBUG
SQL Injection	Critical	webgoat-lessons/.../Assignment5.java:67	DEBUG

DETAILS | STACKTRACE | INPUT | DESCRIPTION | USEFUL LINKS

Method: POST Uri: /challenge/5?username_login=Larry&password_login=%27z
Content type: text_html
Body: =

Cross Site Scripting | High | POST /WebGoat/CrossSiteScripting/phone-home-xss

Merge branch 'master' into fuzzing_challenge5 | 930b628
Update .gitignore | Verified | ed0fbba

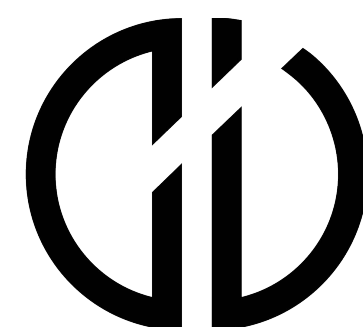
cifuzz bot commented on Jan 21

Found "SQL Injection" while fuzzing: [View Finding](#)

For more information, check [Code Intelligence Fuzzing Academy](#)

cifuzz bot commented on Jan 21

File	Coverage
webgoat-server/src/main/java/org/owasp/webgoat	
HSQLDBDatabaseConfig.java	100%
StartWebGoat.java	100%
webgoat-lessons/secure-passwords/src/main/java/org/owasp/webgoat/secure_password	
SecurePasswordsAssignment.java	100%
SecurePasswords.java	100%



code-intelligence.com