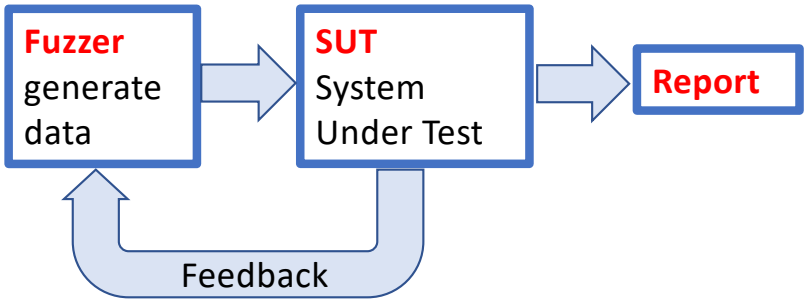
Fuzzing for Test Automation.



Fuzzing or fuzz testing - test software by feeding it with random data – was done since 1950s.

The term "fuzzing" originates from a 1988 class project (fuzz testing unix utilities) taught by Barton Miller at the University of Wisconsin.

Prof. Barton P. Miller
Univ. of Wisconsin

- https://en.wikipedia.org/wiki/Fuzzing
- https://www.guru99.com/fuzz-testing.html

- 2012 – **ClusterFuzz** (Google cloud-based fuzzing infrastructure)
- 2014 – **fuzzer AFL success** in disclosing **Sellchock bugs in Unix Bash shell**
- 2014-15 – fuzzer AFL disclosed the **Heartbleed vulnerability in OpenSSL**
- 2016 – **DARPA (Defense Advanced Research Projects Agency)** competition for developing automatic defense systems that can discover, exploit, and correct software flaws in real-time – used fuzzing as offense strategy to discover flaws in the software of the opponents.
- 2016 - Project Springfield - **Microsoft cloud-based fuzz testing service** for finding security critical bugs in software.
- 2016 - **OSS-Fuzz (Google)** - continuous fuzzing of security-critical open-source projects.
- 2018 - **At Black Hat 2018**, Christopher Domas demonstrated the use of fuzzing to expose the existence of a **hidden RISC core in a processo**r. This core was able to bypass existing security checks to execute Ring 0 commands from Ring 3.

- Fuzzy testing known to find the most serious security faults or defects.
- Fuzzing is one of the most common method hackers used to find vulnerability of the system.

Types of Fuzzes:

Fuzz Testing — A Primer:
- https://medium.com/@priyankasomrah/fuzz-testing-a-primer-2f8f4dc1062

generation-based fuzzer - input data generated from scratch

mutation-based fuzzer -input data generated by modifying existing data – and then select the best inputs (test suite reduction)

dumb fuzzer – not aware of input structure

smart fuzzer – aware of structure of input data. For example, a smart generation-based fuzzer takes the input model or protocol that was provided by the user to generate new inputs.

black-box fuzzer – doesn't know anything about program structure, thus generally can not guarantee good code coverage for testing. But tests are fast and can be done in parallel.

white fuzzer – analyzes program structure. May use symbolic execution to systematically explore different paths in the program. Much slower than black-box fuzzer

grey fuzzer – in-between white and black-box. Use lightweight instrumentation to trace basic block transitions. This leads to a reasonable performance overhead, while informing the fuzzer about the increase in code coverage during fuzzing. This makes gray-box fuzzers extremely efficient. Examples - **AFL** and **libFuzzer** .

This page walks you through setting up
coverage guided fuzzing using **libFuzzer** or **AFL**:
- https://google.github.io/clusterfuzz/setting-up-fuzzing/libfuzzer-and-afl/

The idea is to **insert hooks into execution**.
Easiest method to achieve this is to compile source code using provided compiler to insert hooks.
But it is also possible to decompile an executable file, or to target the LLVM bitcode.

AFL = American Fuzzy Lop (lop := something that is lopped (cut-off), such as branches and twigs lopped off trees)
- http://lcamtuf.coredump.cx/afl/ , read excellent intro here: - https://github.com/google/AFL
Also QuickStartGuide here: - https://github.com/google/AFL/blob/master/docs/QuickStartGuide.txt
AFL is a security-oriented fuzzer originally developed at ~2013 by Michal Zalewski at Google: lcamtuf@google.com.
Usage: compile source code of your system using **AFL** c-compiler (gcc of clang).
This inserts hooks into the code, thus allowing to detect place in the code where things happen, and to do adjustments using feedback.

libFuzzer
- https://llvm.org/docs/LibFuzzer.html
- https://github.com/google/fuzzing/blob/master/tutorial/libFuzzerTutorial.md
- https://scholar.google.com/scholar?start=10&q=fuzz+testing+libfuzzer&hl=en&as_sdt=0,33
- Google for libfuzzer to get more links
libFuzzer is part of the LLVM compiler infrastructure project and comes built-in with the clang compiler.
So you compile your source code with it, for example:

```
clang -g -O1 -fsanitize=fuzzer mytarget.c
```

libFuzzer requires a bit more work to setup (comparing with AFL),
but it is ideal when it comes to fuzzing specific API calls in a library
as well as it has better support for different sanitisers.
Also It is possible to use **AFL** engine while using a target function like in libFuzzer.

Awesome Fuzzing - a curated list of fuzzing resources (books, courses, videos, tools, tutorials, vulnerable applications to practice on)
- https://github.com/secfigo/Awesome-Fuzzing

NEUZZ: Efficient Fuzzing with Neural Program Smoothing (July 2019, **Columbia University** - by Dongdong She, Kexin Pei, Dave Epstein, Junfeng Yang, Baishakhi Ray, and Suman Jana). NEUZZ significantly outperforms 10 state-of-the-art graybox fuzzers on 10 popular real-world programs both at finding new bugs and achieving higher edge coverage.
- https://arxiv.org/pdf/1807.05620.pdf

KLUZZER: Whitebox Fuzzing on Top of LLVM - Targeting LLVM bitcode , easy to combine with libFuzzer. **We need to buy this PDF (~$80**)
- https://link.springer.com/chapter/10.1007/978-3-030-31784-3_14
- https://gitlab.informatik.uni-bremen.de/mhle/kluzzer_atva19

Pulling JPEGs out of thin air - demonstration of the capabilities of AFL:
- https://lcamtuf.blogspot.com/2014/11/pulling-jpegs-out-of-thin-air.html?m=1

Hypothesis - Python library for creating unit tests: https://hypothesis.readthedocs.io/en/latest/index.html

Symbolic Execution (also symbolic evaluation) - a means of analyzing a program to determine what inputs cause each part of a program to execute: https://en.m.wikipedia.org/wiki/Symbolic_execution

Dynamic program analysis – executing with sufficient test inputs to cover almost all possible outputs
- https://en.m.wikipedia.org/wiki/Dynamic_program_analysis

ClusterFuzz – open-sourced by Google in 2019, was used on 25,000 servers to find 16,000 bugs in Chome and 11,000 bugs in other Open Source projects: https://jaxenter.com/clusterfuzz-open-sourced-155414.html

Evolutionary Fuzzing:

Evolutionary fuzzing - converging towards the discovery of weaknesses.
It uses genetic algorithms in order to produce successive generations of test cases populations.

Evolutionary fuzzing - use feedback from each test case to learn the format of the input over time.
For example, by measuring the code coverage of each test case, the fuzzer can work out which properties of the test case exercise a given area of code, and gradually evolve a set of test cases that cover the majority of the program code.

Evolutionary fuzzing often relies on other techniques similar to genetic algorithms and may require some form of binary instrumentation to operate correctly.

libFuzzer – a library for coverage-guided fuzz testing. **LibFuzzer** is in-process, coverage-guided, evolutionary fuzzing engine. **LibFuzzer** is linked with the library under test, and feeds fuzzed inputs to the library via a specific fuzzing entrypoint (aka "target function"); the fuzzer then tracks which areas of the code are reached, and generates mutations on the corpus of input data in order to maximize the code coverage. The code coverage information for libFuzzer is provided by LLVM's SanitizerCoverage instrumentation.
- https://llvm.org/docs/LibFuzzer.html

V-Fuzz: Vulnerability-Oriented Evolutionary Fuzzing
Yuwei Li, Shouling Ji, Chenyang Lv, Yuan Chen, Jianhai Chen, Qinchen Gu, and Chunming Wu
- https://arxiv.org/pdf/1901.01142.pdf

Google Brain TensorFuzz Debugs Neural Networks with Coverage-Guided Fuzzing (CGF) (Odena & Goodfellow, 2018)

Augustus Odena & Ian Goodfellow

https://medium.com/syncedreview/google-brain-tensorfuzz-debugs-neural-networks-with-coverage-guided-fuzzing-89843346fcda

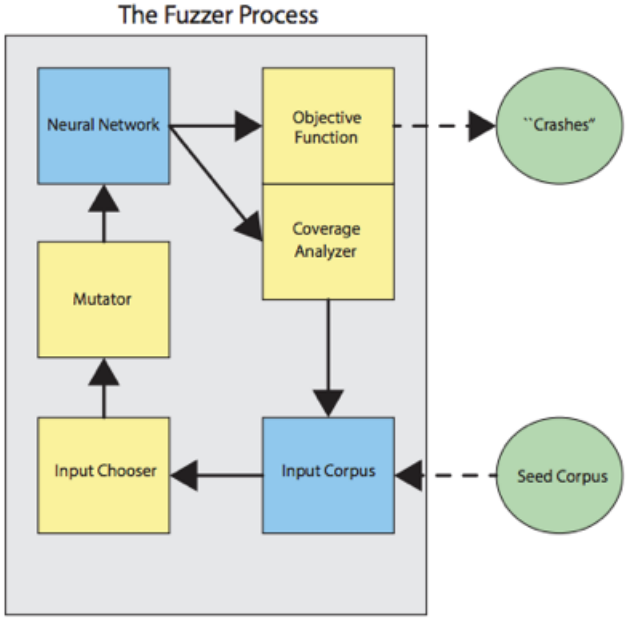
http://proceedings.mlr.press/v97/odena19a/odena19a.pdf

https://github.com/brain-research/tensorfuzz

Fuzzing over TensorFlow graph.

The **TensorFuzz** tool
feeds inputs to an arbitrary TensorFlow graph
and measures coverage by looking at the “activations” of the computation graph.

In coverage-guided fuzzing,
random mutations of inputs to a neural network
are guided by a coverage metric
toward the goal of satisfying user-specified constraints.



Some Fuzz Testing Frameworks

(web applications and cybersecurity):

- **Radamsa** - designed to be easy to use and flexible. It attempts to "just work" for a variety of input types and contains a number of different fuzzing algorithms for mutation.
- **Sulley** - provides a comprehensive generation framework, allowing structured data to be represented for generation-based fuzzing. It also contains components to help with recording test cases and detecting crashes.
- **Peach** - framework can perform smart fuzzing for file formats and network protocols. It can perform both generation- and mutation-based fuzzing and contains components to help with modelling and monitoring the target.
- **SPIKE** - a network protocol fuzzer. It requires good knowledge of C to use and is designed to run on Linux.
- **Grinder** - a web browser fuzzer, which also has features to help in managing large numbers of crashes.
- **NodeFuzz** - a node.js-based harness for web browsers, which includes instrumentation modules to gain further information from the client side.
- **Burp Suite** – Cybersecurity Software
- **Webscarab** – Java framework, http & https, can be used as intercepting proxy
- **OWASP WSFuzzer** – python, for HTTP based SOAP services
- Software testing at Facebook and Google in 2018 - https://screenster.io/software-testing-facebook-google/

Facebook Testing Tools

- Software testing at Facebook and Google in 2018 - https://screenster.io/software-testing-facebook-google/
- **Sapienz** - Facebook acquired Sapienz in 2017, a search-based dynamic code analyzer for Android, relies on fuzz testing, creates models of the system under tests, and generates test sequences. Whereas most dynamic analyzers have to comb through ~15,000 actions to find a crashing bug, Sapienz only needs 150–200 interactions. - https://thenewstack.io/facebooks-tool-for-automated-testing-at-2-billion-users-scale/
- **Infer** - A static code analyzer, Infer scans through code without running it. It detects bugs like memory leaks and null points exceptions in iOS and Android. Facebook bought the technology in 2013 and open-sourced it in 2015. Today, Infer is integral to the testing of Facebook, Instagram, WhatsApp, Messenger, as well as Spotify and Uber. Mozilla, Sky, and Marks & Spencer are among other well-known brands using working with this tool.
- **BrowserLab** - granular monitoring of browser rendering speeds, detects performance regressions.
- **Jest** - Marketed as a framework for JavaScript testing (React, etc.).

More Resources

- A systematic review of fuzzing based on machine learning techniques
  by Yan Wanga, Peng Jiaa, Luping Liub, Jiayong Liua
  - https://arxiv.org/pdf/1908.01262.pdf
- NeuFuzz: Efficient Fuzzing With Deep Neural Network
  by Yunchao Wang; Zehui Wu; Qiang Wei; Qingxian Wang
  - https://ieeexplore.ieee.org/document/8672949
- Microsoft - Neural fuzzing: applying Deep Neural Network (DNN) to software security testing
  … Overall, using neural fuzzing outperformed traditional AFL in most cases.
  - https://www.microsoft.com/en-us/research/blog/neural-fuzzing/
- Coverage-guided Fuzzing for Feedforward Neural Networks
  by Xiaofei Xie; Hongxu Chen; Yi Li; Lei Ma; Yang Liu; Jianjun Zhao - Singapore & Japan
  - https://www.ntu.edu.sg/home/yi_li/files/ase19-deephunter.pdf
- ReluDiff: Differential Verification of Deep Neural Networks
  - by Brandon Paulsen; Jingbo Wang; Chao Wang - https://arxiv.org/pdf/2001.03662.pdf.
- FuzzerGym: A Competitive Framework for Fuzzing and Learning
  - by William Drozd & Michael D. Wagner, CMU - https://arxiv.org/pdf/1807.07490.pdf
- A systematic review of fuzzing based on machine learning techniques
  - by Yan Wanga, Peng Jiaa, Luping Liub, Jiayong Liua, China - https://arxiv.org/pdf/1908.01262.pdf
- The Art, Science, and Engineering of Fuzzing: A Survey (2019)
  - by Valentin J.M. Man`es, HyungSeok Han, Choongwoo Han, Sang Kil Cha, Manuel Egele, Edward J. Schwartz, and Maverick Woo
  - https://arxiv.org/pdf/1812.00140.pdf
- EnFuzz: Ensemble Fuzzing with Seed Synchronization among Diverse Fuzzers - https://arxiv.org/pdf/1807.00182.pdf
- Klee LLVM Execution Engine - http://klee.github.io/
- Microsoft Fuzzing as a service - https://www.microsoft.com/en-us/security-risk-detection/