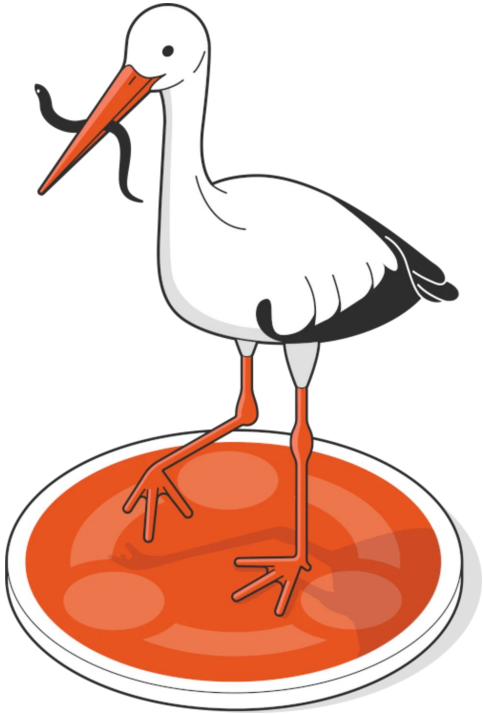


Fuzzing in the open

Integrate your project in OSS-Fuzz for continuous fuzzing

First things first:

Do you have the setup?

If not, install Python 3 + Docker, and run:

```
$ git clone https://github.com/google/oss-fuzz
$ git clone http://github.com/iosifache/fuzzingintheopen
```

The workshop in a nutshell

Fuzzing 101

OSS-Fuzz

Discovering Heartbleed with fuzzing

Tales from a real-life integration

OSS-Fuzz-Gen

Dongge Liu

Working on OSS-Fuzz-Gen

Maintaining OSS-Fuzz and FuzzBench

Research interests in cybersecurity and machine learning

Andrei Iosif

Security engineer @ Snap Inc.

GSoC mentorship, startup advisorship, coffee, and outdoor sports

Ex-member of the Ubuntu Security Team

Jiongchi Yu

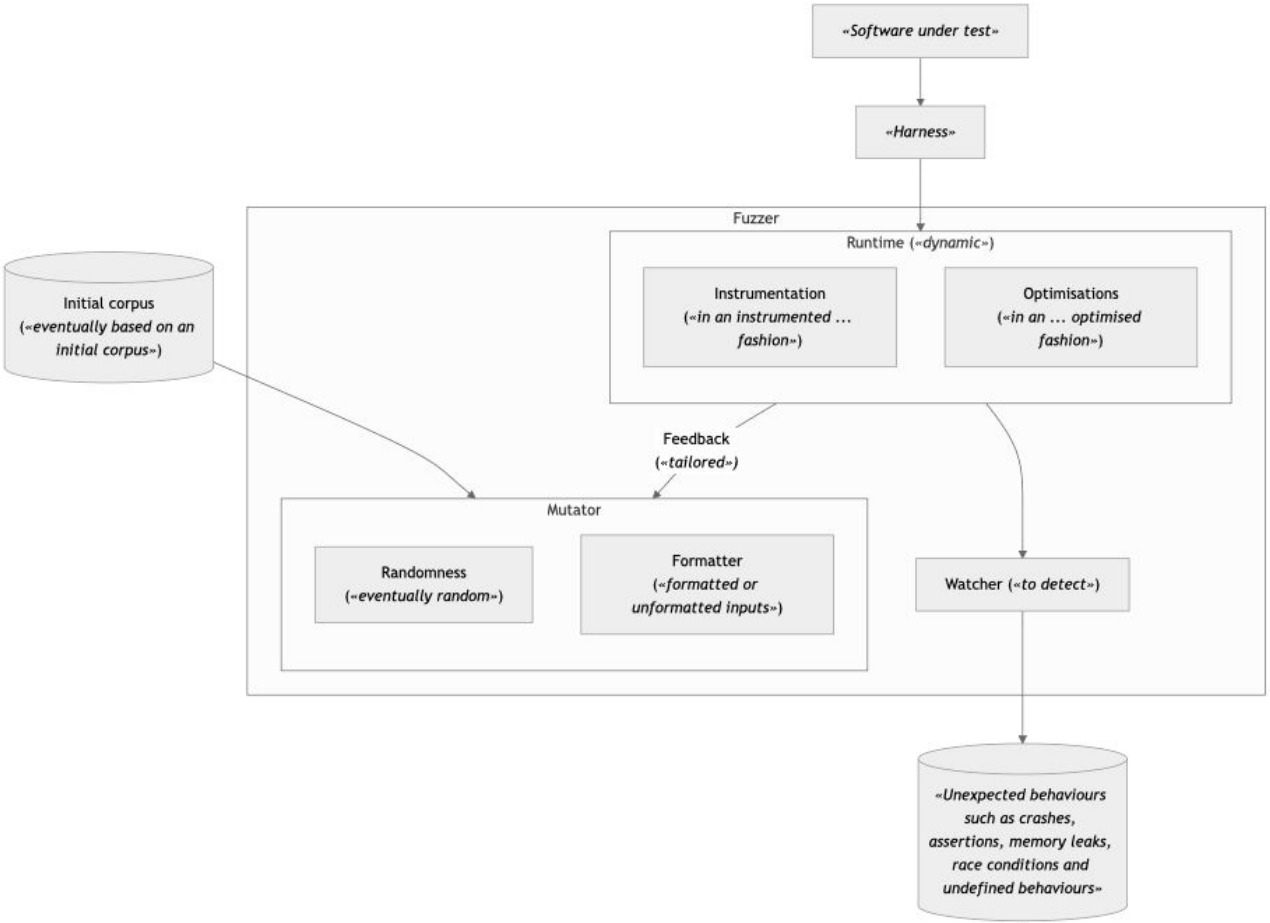
PhD candidate in Computer Science at SMU

Working on integrating OpenPrinting projects into OSS-Fuzz (through GSoC)

Research in cloud native testing and security

Fuzzing

A dynamic software testing mechanism that runs (in an instrumented and optimised fashion) the software under test (through its harness) with tailored (eventually random) (formatted or unformatted) (eventually based on an initial corpus) inputs to detect unexpected behaviours such as crashes, assertions, memory leaks, race conditions and undefined behaviours


«Software under test»
«Harness»
Fuzzer
Runtime («dynamic»)
Instrumentation («in an instrumented ... fashion»)
Optimisations («in an ... optimised fashion»)
Initial corpus («eventually based on an initial corpus»)
Feedback («tailored»)
Mutator
Randomness («eventually random»)
Formatter («formatted or unformatted inputs»)
Watcher («to detect»)
«Unexpected behaviours such as crashes, assertions, memory leaks, race conditions and undefined behaviours»


Valid and accepted inputs
Invalid and rejected inputs


Unexpected input

OSS-Fuzz from Google

- Continuous fuzzing infrastructure for critical open source projects
- Free
- OSS-Fuzz has helped identify and fix over 10,000 vulnerabilities and 36,000 bugs across 1,000 projects
- Supports:
  - Multiple fuzzers: libFuzzer, AFL++, Honggfuzz, and Centipede
  - Multiple languages: C, C++, Go, Java, Javascript, Python, Rust, and Swift
  - CPU architectures: x86_64, i386

OSS-Fuzz Rewards

- Some contributions are rewarded (up to 15k) by Google:
  - Integrations: initial or ideal
  - Coverage: line coverage, FuzzIntrospector, or FuzzBench
  - Vulnerabilities: new sanitisers or finding impactful vulnerabilities

OpenSSL

Cryptography and SSL/TLS Toolkit

The Heartbleed Bug

The Heartbleed Bug is a serious vulnerability in the popular Ope weakness allows stealing the information protected, under normal condition ecure the Internet. SSL/TLS provides communication security and privacy ov as web, email, instant messaging (IM) and some virtual private networks (VPNs).

The Heartbleed bug allows anyone on the Internet to read the memo f by the vulnerable versions of the OpenSSL software. This compromises the secret key se rvice providers and to encrypt the traffic, the names and passwords of the users and the actual c ent. ws attackers to eavesdrop on communications, steal data directly from the services and users and to im onate se es and users.

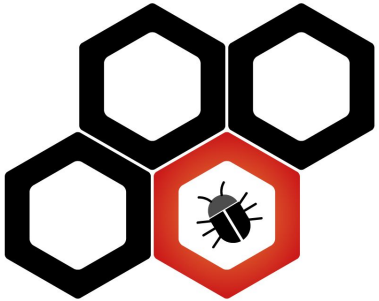
Google Testing Blog

Announcing OSS-Fuzz: Continuous Fuzzing for Open Source Software

Thursday, December 01, 2016

By Mike Aizatsky, Kostya Serebryany (Software Engineers, Dynamic Tools); Oliver Chang, Abhishek Arya (Security Engineers, Google Chrome); and Meredith Whittaker (Open Research Lead).

We are happy to announce OSS-Fuzz, a new Beta program developed over the past years with the Core Infrastructure Initiative community. This program will provide continuous fuzzing for select core open source software.



Assuming you are

The initial founding members were **Mark Cox, Ralf Engelschall, Stephen Henson, Ben Laurie, and Paul Sutton**. In 2018 OpenSSL version numbering skipped from 1.1.

Wikipedia
https://en.wikipedia.org › wiki › OpenSSL

OpenSSL - Wikipedia

, use OSS-Fuzz to detect

*Some steps are infeasible for a 1.5-hours workshop.

PRACTICAL vs. PLACEHOLDER

practical steps of the OSS-Fuzz integration vs. just-described steps (blocked by external factors)


HEARTBLEED

Initialising the project

PRACTICAL

```
host $ cd oss-fuzz

host $ export PROJECT_NAME=openssl-heartbleed

host $ export LANGUAGE=c++

host $ python infra/helper.py generate \
    $PROJECT_NAME \
    --language=$LANGUAGE
host $ cd projects/$PROJECT_NAME
host $ ls
```

Generating SSL certificates

PLACEHOLDER

```
host $ openssl req -x509 \
    -newkey rsa:512 \
    -keyout server.key \
    -out server.pem \
    -days 9999 \
    -subj /CN=a/ \
    -nodes
```

Filling up `project.yaml`

PRACTICAL

```
host $ cat project.yaml

host $ mv ../../../fuzzingintheopen/infra/project.yaml .

host $ cat project.yaml
```

Filling up `Dockerfile`

PRACTICAL

```
host $ cat Dockerfile

host $ mv ../../../fuzzingintheopen/infra/Dockerfile .

host $ cat Dockerfile
```

Filling up `build.sh`

PRACTICAL

```
host $ cat build.sh

host $ mv ../../../fuzzingintheopen/infra/build.sh .

host $ cat build.sh
```

Tips

- Don’t overwrite `CFLAGS` or `CXXFLAGS`.

Creating a harness in `target.cc`

PRACTICAL

```
host $ mv ../../../fuzzingintheopen/infra/runtime .

host $ ls runtime

host $ mv ../../../fuzzingintheopen/infra/target.cc .

host $ cat target.cc
```

Tips

- Maintain the fuzz target `target.cc` in the project-under-test repository

Verifying the files’ correctness

PRACTICAL

```
host $ docker build -t fuzzingintheopen .
host $ docker run -it -v "$(pwd)/out":/out fuzzingintheopen /bin/bash
docker $ /src/build.sh
docker $ /out/openssl-fuzzer
```

Building the fuzzers

PRACTICAL

```
host $ cd ../../

host $ python infra/helper.py build_fuzzers \

    --sanitizer address \

    --engine libfuzzer \

    --architecture <x86_64|i386|aarch64> \

    $PROJECT_NAME
```

Running the fuzzer (and profit!)

PRACTICAL

```
host $ export FUZZER_NAME=openssl-fuzzer
host $ python infra/helper.py run_fuzzer \
    --sanitizer address \
    --engine libfuzzer \
    --architecture <x86_64|i386|aarch64> \
    $PROJECT_NAME \
    $FUZZER_NAME
host $ xxd ./build/out/heartbleed/crash-*
```

Validating the crash

PRACTICAL

```
host $ python infra/helper.py reproduce \
    $PROJECT_NAME \
    $FUZZER_NAME \
    <crash_file>
host $ ./build/out/openssl-fuzzer <crash_file>
```

Hosting the harness in OSS-Fuzz

PLACEHOLDER

Hosting the harness in OSS-Fuzz

We'll review the fuzz folder from the OpenSSL repository:

https://github.com/openssl/openssl/tree/master/fuzz

Tip: Don't place the harnesses in the OSS-Fuzz repository.

Having a PR accepted inside the OSS-Fuzz repo

PLACEHOLDER

Having a PR accepted inside the OSS-Fuzz repo

The initial integration PR of project Pacemaker is a good example:

1. Describe Project purpose and importance.
2. List major users.
   - Highlight critical role of the project.
3. Link maintainer's approval and fuzz target PR.

Tip: Addressing these points helps our panel decide on accepting your integration.

Checking the remote build logs

PLACEHOLDER

Checking the remote build logs

The latest build logs are linked in Fuzz Introspector's UI:

https://introspector.oss-fuzz.com/project-profile?project=openssl

They can also fail, as illustrated here:

https://issues.oss-fuzz.com/issues/42533795

Try to spot the issue in the logs.

Checking the Fuzz Introspector page

PLACEHOLDER

Checking the Fuzz Introspector page

Access the Fuzz Introspector page of OpenSSL:

https://introspector.oss-fuzz.com/project-profile?project=openssl

What's the line coverage of the project? What functions can be fuzzer to increase the coverage?

Receiving a mail with a crash

PLACEHOLDER

Receiving a mail with a crash

Use https://issues.oss-fuzz.com/issues/42535311 as an example.

Patching

PRACTICAL

Patching

1. Inspect the `infra/heartbleed.patch` Git patch.
2. Modify `Dockerfile` to apply the patch after cloning the `openssl` repository.
3. Rebuild.
4. Fuzz again with OSS-Fuzz.

A success story: GSoC x OpenPrinting x OSS-Fuzz

- GSoC has helped in securing OpenPrinting with integrating unit testing and CI testing of OpenPrinting projects, e.g. OpenPrinting/libcupsfilters#58
- Previous OSS-Fuzz integrations of OSS projects during GSoC: nginx, PostgreSQL, and Envoy Proxy in 2020

OpenPrinting

- OpenPrinting is an open source organisation focused on standardizing printing support for Linux and Unix-like systems by providing drivers, filters, and printing tools.
- 20 out of 37 OpenPrinting projects are mainly implemented in memory unsafe languages (C or C++).
- OpenPrinting projects lacks sufficient fuzz testing.

OpenPrinting Security

- Many widely used software is developed based on OpenPrinting projects, e.g, GTK, Gnome, and LibreOffice.
- Projects from OpenPrinting are pre-installed in Ubuntu releases.
- Security issues in OpenPrinting projects can cause severe consequences, for example CVE-2024-47175, CVE-2024-47176, CVE-2024-47176, and CVE-2024-47177, which leads to the remote code execution (RCE).

Integrating OpenPrinting Projects into OSS-Fuzz

- Three main projects in OpenPrinting
  - `cups`
  - `libcups`
  - `cups-filters`
- Pending projects
  - `cups-filters`
  - `libcupsfilters`
  - `cups-browsed`

Integrating OpenPrinting Projects into OSS-Fuzz

- Our progress
  - 41 issues reported with 21 resolved
  - 5000+ LoC changes
  - Enabling Fuzz Introspector to support LLM-assisted fuzz driver generation (via OSS-Fuzz-Gen).
- Advanced fuzz integration
  - From unit testing to fuzz harness
  - Adapting various of test input format (e.g., IPP Protocol, PDF file, and `argv` input)
  - Feedback from Fuzz Introspector to OSS-Fuzz-Gen
  - Refer to our repo: https://github.com/OpenPrinting/fuzzing

OSS-Fuzz-Gen

- Given a function, automatically generates fuzz targets.

	Human-written	**OSS-Fuzz-Gen**
Fuzzing knowledge	Some required	LLM pretrained
Project domain knowledge	Required	Programmatically supplied in prompts
Time	Hours-Days	Seconds
Cost	$1000+	$0.1-

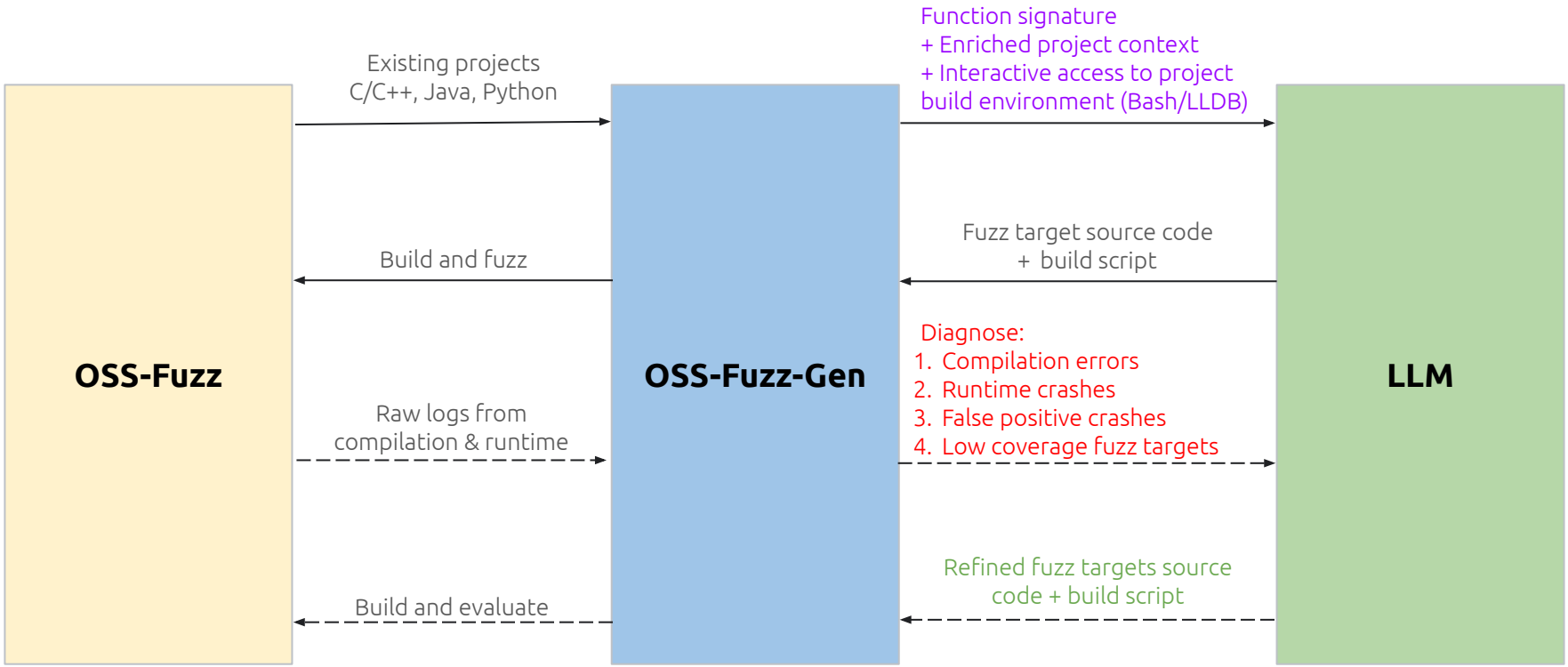
OSS-Fuzz-Gen Trophies

- Uncovered 26 new bugs (e.g., CVE-2024-9143) in projects were previously thoroughly fuzzed with tons of CPU hours
- Contact us to apply OSS-Fuzz-Gen to a new project or apply your ideas to it.

Project	Bug Type
cJSON	OOB read
libplist	OOB read
hunspell	OOB read
zstd	OOB write
gdbm	Stack buffer underflow
hoextdown	Use of uninitialised memory
pjsip	OOB read
pjsip	OOB read
gpac	OOB read
gpac	OOB read/write
gpac	OOB read
gpac	OOB read
sqlite3	OOB read

Project	Bug Type
htslib	OOB read
libical	OOB read
croaring	OOB read
openssl	OOB read/write
Undisclosed	OOB read
Undisclosed	OOB read
Undisclosed	OOB read
Undisclosed	OOB read
Undisclosed	OOB read
Undisclosed	Use after free
Undisclosed	Use of uninitialised memory
Undisclosed	Java RCE
Undisclosed	Java Regexp DoS

OSS-Fuzz-Gen Overview



The workshop in a nutshell

Fuzzing 101

OSS-Fuzz

Discovering Heartbleed with fuzzing

Tales from a real-life integration

OSS-Fuzz-Gen


That's all Folks!