/ossfuzz

Source: `~/.claude/skills/tob-testing-handbook-skills/skills/ossfuzz/SKILL.md`

---

name: ossfuzz type: technique description: > OSS-Fuzz provides free continuous fuzzing for open source projects. Use when setting up continuous fuzzing infrastructure or enrolling projects.

OSS-Fuzz

OSS-Fuzz is an open-source project developed by Google that provides free distributed infrastructure for continuous fuzz testing. It streamlines the fuzzing process and facilitates simpler modifications. While only select projects are accepted into OSS-Fuzz, the project's core is open-source, allowing anyone to host their own instance for private projects.

Overview

OSS-Fuzz provides a simple CLI framework for building and starting harnesses or calculating their coverage. Additionally, OSS-Fuzz can be used as a service that hosts static web pages generated

from fuzzing outputs such as coverage information.

Key Concepts

Concept	Description
helper.py	CLI script for building images, building fuzzers, and running harnesses locally
Base Images	Hierarchical Docker images providing build dependencies and compilers
project.yaml	Configuration file defining project metadata for OSS-Fuzz enrollment
Dockerfile	Project-specific image with build dependencies
build.sh	Script that builds fuzzing harnesses for your project
Criticality Score	Metric used by OSS-Fuzz team to evaluate project acceptance

When to Apply

Apply this technique when:

- Setting up continuous fuzzing for an open-source project
- Need distributed fuzzing infrastructure without managing servers
- Want coverage reports and bug tracking integrated with fuzzing
- Testing existing OSS-Fuzz harnesses locally
- Reproducing crashes from OSS-Fuzz bug reports

Skip this technique when:

- Project is closed-source (unless hosting your own OSS-Fuzz instance)
- Project doesn't meet OSS-Fuzz's criticality score threshold
- Need proprietary or specialized fuzzing infrastructure
- Fuzzing simple scripts that don't warrant infrastructure

Quick Reference

Task	Command
Clone OSS-Fuzz	`git clone https://github.com/google/oss-fuzz`
Build project image	`python3 infra/helper.py build_image --pull <project>`

Task	Command
Build fuzzers with ASan	`python3 infra/helper.py build_fuzzers --sanitizer=address <project>`
Run specific harness	`python3 infra/helper.py run_fuzzer <project> <harness>`
Generate coverage report	`python3 infra/helper.py coverage <project>`
Check helper.py options	`python3 infra/helper.py --help`

OSS-Fuzz Project Components

OSS-Fuzz provides several publicly available tools and web interfaces:

Bug Tracker

The bug tracker allows you to:

- Check bugs from specific projects (initially visible only to maintainers, later made public)
- Create new issues and comment on existing ones
- Search for similar bugs across **all projects** to understand issues

Build Status System

The build status system helps track:

- Build statuses of all included projects
- Date of last successful build
- Build failures and their duration

Fuzz Introspector

Fuzz Introspector displays:

- Coverage data for projects enrolled in OSS-Fuzz
- Hit frequency for covered code
- Performance analysis and blocker identification

Read this case study for examples and explanations.

Step-by-Step: Running a Single Harness

You don't need to host the whole OSS-Fuzz platform to use it. The helper script makes it easy to run individual harnesses locally.

Step 1: Clone OSS-Fuzz

```
git clone https://github.com/google/oss-fuzz
cd oss-fuzz
python3 infra/helper.py --help
```

Step 2: Build Project Image

```
python3 infra/helper.py build_image --pull <project-name>
```

This downloads and builds the base Docker image for the project.

Step 3: Build Fuzzers with Sanitizers

```
python3 infra/helper.py build_fuzzers --sanitizer=address <project-name>
```

Sanitizer options:

- `--sanitizer=address` for AddressSanitizer with LeakSanitizer
- Other sanitizers available (language support varies)

Note: Fuzzers are built to `/build/out/<project-name>/` containing the harness executables, dictionaries, corpus, and crash files.

Step 4: Run the Fuzzer

```
python3 infra/helper.py run_fuzzer <project-name> <harness-name> [<fuzzer-args>]
```

The helper script automatically runs any missed steps if you skip them.

Step 5: Coverage Analysis (Optional)

First, install gsutil (skip gcloud initialization).

```
python3 infra/helper.py build_fuzzers --sanitizer=coverage <project-name>
python3 infra/helper.py coverage <project-name>
```

Use `--no-corpus-download` to use only local corpus. The command generates and hosts a coverage report locally.

See official OSS-Fuzz documentation for details.

Common Patterns

Pattern: Running irssi Example

Use Case: Testing OSS-Fuzz setup with a simple enrolled project

```
# Clone and navigate to OSS-Fuzz
git clone https://github.com/google/oss-fuzz
cd oss-fuzz

# Build and run irssi fuzzer
python3 infra/helper.py build_image --pull irssi
python3 infra/helper.py build_fuzzers --sanitizer=address irssi
python3 infra/helper.py run_fuzzer irssi irssi-fuzz
```

Expected Output:

```
INFO:__main__:Running: docker run --rm --privileged --shm-size=2g --platform linux/amd64 -i -e
FUZZING_ENGINE=libfuzzer -e SANITIZER=address -e RUN_FUZZER_MODE=interactive -e HELPER=True -v
/private/tmp/oss-fuzz/build/out/irssi:/out -t gcr.io/oss-fuzz-base/base-runner run_fuzzer
irssi-fuzz.
Using seed corpus: irssi-fuzz_seed_corpus.zip
/out/irssi-fuzz -rss_limit_mb=2560 -timeout=25 /tmp/irssi-fuzz_corpus -max_len=2048 <
/dev/null
INFO: Running with entropic power schedule (0xFF, 100).
INFO: Seed: 1531341664
INFO: Loaded 1 modules   (95687 inline 8-bit counters): 95687 [0x1096c80, 0x10ae247),
INFO: Loaded 1 PC tables (95687 PCs): 95687 [0x10ae248,0x1223eb8),
INFO:      719 files found in /tmp/irssi-fuzz_corpus
```

```
INFO: seed corpus: files: 719 min: 1b max: 170106b total: 367969b rss: 48Mb
#720        INITED cov: 409 ft: 1738 corp: 640/163Kb exec/s: 0 rss: 62Mb
#762        REDUCE cov: 409 ft: 1738 corp: 640/163Kb lim: 2048 exec/s: 0 rss: 63Mb L: 236/2048
MS: 2 ShuffleBytes-EraseBytes-
```

Pattern: Enrolling a New Project

Use Case: Adding your project to OSS-Fuzz (or private instance)

Create three files in `projects/<your-project>/`:

1. project.yaml - Project metadata:

```
homepage: "https://github.com/yourorg/yourproject"
language: c++
primary_contact: "your-email@example.com"
main_repo: "https://github.com/yourorg/yourproject"
fuzzing_engines:
  - libfuzzer
sanitizers:
  - address
  - undefined
```

2. Dockerfile - Build dependencies:

```
FROM gcr.io/oss-fuzz-base/base-builder
RUN apt-get update && apt-get install -y \
    autoconf \
    automake \
    libtool \
    pkg-config
RUN git clone --depth 1 https://github.com/yourorg/yourproject
WORKDIR yourproject
COPY build.sh $SRC/
```

3. build.sh - Build harnesses:

```
#!/bin/bash -eu
./autogen.sh
./configure --disable-shared
```

```
make -j$(nproc)

# Build harnesses
$CXX $CXXFLAGS -std=c++11 -I. \
    $SRC/yourproject/fuzz/harness.cc -o $OUT/harness \
    $LIB_FUZZING_ENGINE ./libyourproject.a

# Copy corpus and dictionary if available
cp $SRC/yourproject/fuzz/corpus.zip $OUT/harness_seed_corpus.zip
cp $SRC/yourproject/fuzz/dictionary.dict $OUT/harness.dict
```

Docker Images in OSS-Fuzz

Harnesses are built and executed in Docker containers. All projects share a runner image, but each project has its own build image.

Image Hierarchy

Images build on each other in this sequence:

1. **base_image** - Specific Ubuntu version
2. **base_clang** - Clang compiler; based on `base_image`
3. **base_builder** - Build dependencies; based on `base_clang`
   - Language-specific variants: `base_builder_go`, etc.
   - See /oss-fuzz/infra/base-images/ for full list
4. **Your project Docker image** - Project-specific dependencies; based on `base_builder` or language variant

Runner Images (Used Separately)

- **base_runner** - Executes harnesses; based on `base_clang`
- **base_runner_debug** - With debug tools; based on `base_runner`

Advanced Usage

Tips and Tricks

Tip	Why It Helps
Don't manually copy source code	Project Dockerfile likely already pulls latest version
Check existing projects	Browse oss-fuzz/projects for examples
Keep harnesses in separate repo	Like curl-fuzzer - cleaner organization
Use specific compiler versions	Base images provide consistent build environment
Install dependencies in Dockerfile	May require approval for OSS-Fuzz enrollment

Criticality Score

OSS-Fuzz uses a criticality score to evaluate project acceptance. See this example for how scoring works.

Projects with lower scores may still be added to private OSS-Fuzz instances.

Hosting Your Own Instance

Since OSS-Fuzz is open-source, you can host your own instance for:

- Private projects not eligible for public OSS-Fuzz
- Projects with lower criticality scores
- Custom fuzzing infrastructure needs

Anti-Patterns

Anti-Pattern	Problem	Correct Approach
Manually pulling source in build.sh	Doesn't use latest version	Let Dockerfile handle git clone
Copying code to OSS-Fuzz repo	Hard to maintain, violates separation	Reference external harness repo
Ignoring base image versions	Build inconsistencies	Use provided base images and compilers
Skipping local testing	Wastes CI resources	Use helper.py locally before PR
Not checking build status	Unnoticed build failures	Monitor build status page regularly

Tool-Specific Guidance

libFuzzer

OSS-Fuzz primarily uses libFuzzer as the fuzzing engine for C/C++ projects.

Harness signature:

```
extern "C" int LLVMFuzzerTestOneInput(const uint8_t *data, size_t size) {
    // Your fuzzing logic
    return 0;
}
```

Build in build.sh:

```
$CXX $CXXFLAGS -std=c++11 -I. \
    harness.cc -o $OUT/harness \
    $LIB_FUZZING_ENGINE ./libproject.a
```

Integration tips:

- Use `$LIB_FUZZING_ENGINE` variable provided by OSS-Fuzz
- Include `-fsanitize=fuzzer` is handled automatically
- Link against static libraries when possible

AFL++

OSS-Fuzz supports AFL++ as an alternative fuzzing engine.

Enable in project.yaml:

```
fuzzing_engines:
  - afl
  - libfuzzer
```

Integration tips:

- AFL++ harnesses work alongside libFuzzer harnesses
- Use persistent mode for better performance
- OSS-Fuzz handles engine-specific compilation flags

Atheris (Python)

For Python projects with C extensions.

Example from cbor2 integration:

Harness:

```
import atheris
import sys
import cbor2

@atheris.instrument_func
def TestOneInput(data):
    fdp = atheris.FuzzedDataProvider(data)
    try:
        cbor2.loads(data)
    except (cbor2.CBORDecodeError, ValueError):
        pass

def main():
    atheris.Setup(sys.argv, TestOneInput)
    atheris.Fuzz()

if __name__ == "__main__":
    main()
```

Build in build.sh:

```
pip3 install .
for fuzzer in $(find $SRC -name 'fuzz_*.py'); do
  compile_python_fuzzer $fuzzer
done
```

Integration tips:

- Use `compile_python_fuzzer` helper provided by OSS-Fuzz
- See Continuously Fuzzing Python C Extensions blog post

Rust Projects

Enable in project.yaml:

```
language: rust
fuzzing_engines:
```

```
  - libfuzzer
sanitizers:
  - address  # Only AddressSanitizer supported for Rust
```

Build in build.sh:

```
cargo fuzz build -O --debug-assertions
cp fuzz/target/x86_64-unknown-linux-gnu/release/fuzz_target_1 $OUT/
```

Integration tips:

- Rust supports only AddressSanitizer with libfuzzer
- Use cargo-fuzz for local development
- OSS-Fuzz handles Rust-specific compilation

Troubleshooting

Issue	Cause	Solution
Build fails with missing dependencies	Dependencies not in Dockerfile	Add `apt-get install` or equivalent in Dockerfile
Harness crashes immediately	Missing input validation	Add size checks in harness
Coverage is 0%	Harness not reaching target code	Verify harness actually calls target functions
Build timeout	Complex build process	Optimize build.sh, consider parallel builds
Sanitizer errors in build	Incompatible flags	Use flags provided by OSS-Fuzz environment variables
Cannot find source code	Wrong working directory in Dockerfile	Set WORKDIR or use absolute paths

Related Skills

Tools That Use This Technique

Skill	How It Applies
libfuzzer	Primary fuzzing engine used by OSS-Fuzz
aflpp	Alternative fuzzing engine supported by OSS-Fuzz

Skill	How It Applies
atheris	Used for fuzzing Python projects in OSS-Fuzz
cargo-fuzz	Used for Rust projects in OSS-Fuzz

Related Techniques

Skill	Relationship
coverage-analysis	OSS-Fuzz generates coverage reports via helper.py
address-sanitizer	Default sanitizer for OSS-Fuzz projects
fuzz-harness-writing	Essential for enrolling projects in OSS-Fuzz
corpus-management	OSS-Fuzz maintains corpus for enrolled projects

Resources

Key External Resources

OSS-Fuzz Official Documentation Comprehensive documentation covering enrollment, harness writing, and troubleshooting for the OSS-Fuzz platform.

Getting Started Guide Step-by-step process for enrolling new projects into OSS-Fuzz, including requirements and approval process.

cbor2 OSS-Fuzz Integration PR Real-world example of enrolling a Python project with C extensions into OSS-Fuzz. Shows:

- Initial proposal and project introduction
- Criticality score evaluation
- Complete implementation (project.yaml, Dockerfile, build.sh, harnesses)

Fuzz Introspector Case Studies Examples and explanations of using Fuzz Introspector to analyze coverage and identify fuzzing blockers.

Video Resources

Check OSS-Fuzz documentation for workshop recordings and tutorials on enrollment and harness development.

Revision #5
Created 2026-02-18 08:40:12 UTC by John

Updated 2026-06-21 20:01:28 UTC by John