Fuzzing Lab Assignment

CS412 Software Security Lab 2

Spring Semester 2025

Introduction

Fuzzing has demonstrated its efficiency in discovering vulnerabilities [3] and being integrated into the industry testing pipeline. Unlike testing a standalone program, fuzzing a software system relies on the well-written drivers to interact with exposed library APIs [4] and the automated testing framework [1].

In this assignment, you will learn about how to improve the library harnesses in OSS-Fuzz [1], Google's fuzzing infrastructure. You are supposed to work in groups of up to four people.

In this assignment you will do the following:

- Understand and evaluate the existing fuzzing harnesses
- Identify shortcomings in the existing fuzzing harnesses
- Update the existing fuzzing harnesses or add new fuzzing harnesses to address the identified shortcomings
- Evaluate the updated harnesses
- Triage a crash in the chosen project
- Write a report on all the above points

Your submission.zip can be submitted until the deadline: 8.5.2025 23:59 on Moodle. The submission.zip will contain the coverage reports and scripts to run your changes along with the report.pdf. Please make sure to include the names and SCIPER numbers of all students of your group in the report.pdf. The report also needs to contain the links to the forks of the relevant repositories with your changes. The report should not be longer than 12 pages (double or single column, A4, reasonable font size). To give you an idea of what we expect from you as well as a simple example check the example_report.pdf. Your submission.zip **needs** to adhere to a strict structure that we will outline in **Submission Format section** . Additionally, we expect the scripts you provide to work out of the box. If not you will get **0 points** for the (Repository) graded deliverables.

You will be graded on the following criteria:

- Quality of the report
- Quality of updates to existing fuzzing harnesses
- Reproducibility of your experiments

For each subtask we provide you with the expected deliverables. Deliverables marked with (Repository) are graded pass/fail based on whether or not we are able to run/build your changes ourselves. Deliverables marked with (Report) are graded based on the clarity of the text and the technical correctness of the content. Overall you can achieve 100pts.

You may choose one of the following open source projects to work on:

- Curl: `https://introspector.oss-fuzz.com/project-profile?project=curl`
- Freetype2: `https://introspector.oss-fuzz.com/project-profile?project=freetype2`
- Harfbuzz: `https://introspector.oss-fuzz.com/project-profile?project=harfbuzz`
- Lcms: `https://introspector.oss-fuzz.com/project-profile?project=lcms`
- Libjpeg-turbo: `https://introspector.oss-fuzz.com/project-profile?project=libjpeg-turbo`
- Libpcap: `https://introspector.oss-fuzz.com/project-profile?project=libpcap`
- Libpng: `https://introspector.oss-fuzz.com/project-profile?project=libpng`
- Libxml2: `https://introspector.oss-fuzz.com/project-profile?project=libxml2`
- Libxslt: `https://introspector.oss-fuzz.com/project-profile?project=libxslt`

All the projects are integrated with oss-fuzz [1], Google's open source fuzzing project. Oss-fuzz allows contributors of open source projects to have their project fuzzed continuously on Google's servers. The only requirement is providing adequate harnesses and build scripts. As part of oss-fuzz, each of the above projects ships fuzzing harnesses and build scripts which are deployed on Google's cloud infrastructure. The links point to the project's entry on oss-fuzz introspector [2], a website storing aggregated information on oss-fuzz projects. The website for your chosen project contains the following useful information:

- The link to the oss-fuzz entry in the oss-fuzz repository (containing the relevant build scripts)

- The link to download the coverage report for this project
- The link to the project's repository

However, you can also build and run these fuzzers locally. Which is what you will do in this part. For example building, fuzzing and generating the coverage for <project> and <harness_name>:

```
git clone https://github.com/google/oss-fuzz && cd oss-fuzz
python3 infra/helper.py build_image <project>
python3 infra/helper.py build_fuzzers <project>

#store results to build/out/corpus
mkdir build/out/corpus/
python3 infra/helper.py run_fuzzer <project> <harness_name> --
    corpus-dir build/out/corpus

#generate coverage
python3 infra/helper.py build_fuzzers --sanitizer coverage <project
    >
python3 infra/helper.py coverage <project> --corpus-dir build/out/
    corpus/ --fuzz-target <harness_name>
```

The source of the oss-fuzz scripts will be in the corresponding project folder in the oss-fuzz repository of the format `https://github.com/google/oss-fuzz/tree/master/projects/$project_name`.

Part1: Running Existing Fuzzing Harnesses (20pt)

In the first part of the assignment you will run the existing fuzzing harnesses yourself and apply minor modifications to warm up and get familiar with oss-fuzz.

Clone the project's repository, the oss-fuzz repository and build the fuzzing harnesses. You can look at the oss-fuzz integration to see how the harnesses are built. Afterwards run the harnesses for a bit to convince yourself everything is working as expected.

Evaluating fuzzers is usually done by using code coverage (which part of the code was executed by the fuzzer). For this part of the assignment pick an existing harness and fuzz it for 4 hours. Use the generated seed corpus to generate a coverage report. (Note that fuzzing for 4 hours is quite short, standard practice in academia is at least 24 hours and in the industry fuzzers will run for weeks if not months.)

Modify the build script of your chosen harness such that the fuzzer is started without any initial seeds (pick a harness that has initial seeds; in case all the harnesses do not use a seed corpus, add a seed of your choosing instead). Fuzz for 4 hours and generate the coverage report. Compare the generated coverage with the coverage of the seeded fuzzer. The deliverables for this part are:

- (Repository) A diff file that we can apply to the oss-fuzz repository and/or project repository to run your chosen fuzzing harness without (or with)

initial seeds. Two run scripts to run the harness without your changes and with your changes. A directory with the oss-fuzz generated coverage report obtained by fuzzing with and without your modification for 4 hours. 10pt

- (Report) The commands you used to build and run the existing fuzz drivers with the default corpus and with an empty corpus, location of the diff file and explanation of your changes. 5pt
- (Report) Discuss the difference between the coverage from the runs with empty and default corpus. 5pt

Part 2: Analyzing Existing Fuzzing Harnesses (30pt)

After you've gotten familiar with the existing fuzz drivers, spend some time trying to understand how they work. Look at the **oss-fuzz introspector** report and try to understand why certain code may not be covered by the existing fuzz drivers. The outcome of this part should be to identify two significant uncovered code regions. You should justify why you consider these code regions significant. Think about why an uncovered code region is important.

Furthermore, you should explain why these regions are not covered by the existing fuzz driver. (The seed corpus can also have an influence on what is covered). You will use this information for part 3, where you will enhance the existing fuzz drivers or update the fuzzing seed corpus or write new fuzz drivers to cover the as of yet uncovered code regions identified above. The deliverables for this part are:

- Discussion of two significant **oss-fuzz introspector uncovered** code regions and why the existing fuzzers are unable to cover this code. For each code region:
  - (Report) Justification of significance. 7pt
  - (Report) Explanation of shortcoming of existing harnesses. 8pt

Part 3: Improving the Existing Fuzzers 30pt

In this part of the assignment you will implement improvements to the existing harnesses to cover the code regions identified in part 2. All your changes should be done in forks of the existing project repository and in the oss-fuzz repository. You are free on how you implement your changes. However most likely your changes will involve one or more of the following:

- Update existing harnesses (simply modify the existing fuzz drivers)

- Add new harnesses (add these new files to your forked repos, update the build script to build your new driver and update the oss-fuzz files to run your new harness)
- Update the fuzzing corpus (understand where the seeds are coming from in the existing fuzzing infrastructure and add your own seeds)

We expect you to include two **different changes**, i.e., not simply include two harnesses. We expect changes to the fuzzing infrastructure such that your improved fuzzers are able to cover the previously uncovered regions. For each of these two uncovered regions you need to demonstrate that your changes are now able to cover the previously identified uncovered code region. Note that you do not need to achieve 100% coverage of the previously uncovered regions, but there should be clear improvement to the harnesses that improves the fuzzers ability to explore the previously not fuzzed code. Do this by fuzzing first without your changes for 4 hours for three times and then with your changes again 4 hours for three times, generating the coverage report for both runs and comparing the two. The deliverables for this part are:

- Updates to the fuzzing infrastructure that improve the existing harnesses so the **two** uncovered code regions identified in part 2 can be covered. For **each** uncovered region we expect:
  - (Repository) Implementation of your changes, buildable and runnable by us via the run script along with the diff of the oss-fuzz and project repository. 3pt
  - (Repository) Coverage report after 4 hours of fuzzing with your change (we expect a folder with the coverage html files in your repository). 2pt
  - (Report) Discussion of your change, how to build and run your new harnesses, which code regions is targeted, description of your implementation and the rationale behind your changes. 5pt
  - (Report) Discussion of the difference in line coverage after fuzzing for 4 hours with your newly improved harness compared to the line coverage originally. Discuss further possible improvements for your target code region. 5pt

Part 4: Crash Analysis (20pt)

In this part of the assignment, the goal is to triage a crash found by your improved fuzzer. In case you did not discover a bug, you can reproduce a prior memory corruption crash. If you reproduce an old bug, you need to build a proof of concept that we can run to reproduce the crash. Ideally, you provide another link to a fork of the project repository with the vulnerable version. To triage a crash, understand the root cause of the bug, propose a fix, and justify your fix. The deliverables for this part are:

- (Repository) Script and PoC to reproduce the crash (either old or newly found) 5pt
- (Report) Root cause analysis of the crash and propose a fix 5pt
- (Report) Analyze the severity of the crash (is it likely to be triggerable by an attacker in production). Exploitability analysis: exploitation primitives that may be obtained from this crash 10pt

Submission Format

In this section we give you detailed information on the expected submission. We reiterate that failing to adhere to this structure or failure to run your scripts will result in the (Repository) deliverables of the corresponding Parts to be given **0 points**!. Figure 1 shows you the expected format of your submission.

We expect all the run*.sh scripts to be standalone (not dependent on running other scripts) and expect them to run correctly on a clean Ubuntu 22.04 machine with docker installed. Thus each run script needs to setup your fork of oss-fuzz and the project directory along with building the required artifacts. Furthermore we specify the specific requirements for the run scripts:

- `run.w_corpus.sh`: build target harness with seeds, run for 4 hours and generate coverage report
- `run.w_o_corpus.sh`: build target harness without seeds, run for 4 hours and generate coverage report
- `run.improve1.sh`: build your fuzzer improvement1, run the updated fuzzer for 4 hours, generate the coverage report
- `run.improve2.sh`: build your fuzzer improvement2, run the updated fuzzer for 4 hours, generate the coverage report
- `run.poc.sh`: build the poc that reproduces the crash and runs it

References

[1] Google. oss-fuzz. `https://github.com/google/oss-fuzz`.

[2] Google. oss-fuzz-introspector. `https://introspector.oss-fuzz.com/`.

[3] google. Clusterfuzz. `https://google.github.io/clusterfuzz/`, 2023.

[4] libfuzzer. libfuzzer. `https://llvm.org/docs/LibFuzzer.html`, 2023.

```
> tree submission/
.
+-- part1/
|   +-- report/ # coverage reports
|   |   +-- w_corpus/
|   |   +-- w_o_corpus/
|   |-- oss-fuzz.diff # diff of oss-fuzz repository
|   |-- project.diff # diff of project repository
|   |-- run.w_corpus.sh # script to run campaign with corpus
|   |-- run.w_o_corpus.sh # script to run campaign without corpus
+-- part3/
|   +-- coverage_noimprove/ # coverage report without improvements
|   +-- improve1/ # how to improve region 1
|   |   +-- coverage_improve1/ # coverage report with improvement1
|   |   |-- run.improve1.sh # script to build and run fuzzer
    improvement1
|   |   |-- * # all files necessary to build/run improvement1
|   |   |-- oss-fuzz.diff # diff of oss-fuzz repository
|   |   |-- project.diff # diff of project repository
|   +-- improve2/ # analogous to improve1
|   |   +-- coverage_improve2/
|   |   |-- run.improve2.sh
|   |   |-- *
|   |   |-- oss-fuzz.diff
|   |   |-- project.diff
+-- part4/
|    |-- * # Files necessary to reproduce the bug
|    |-- run.poc.sh # instruction to reproduce the bug
|-- report.pdf # the report pdf
```

Figure 1: The expected content of your submission.zip