CS-412 Software Security Lab 2

Fuzzing Lab Report
Spring Semester 2025

Project: Tmux

Luca Di Bello (SCIPER 367552) Federico Villa (SCIPER XYZ)
Noah El Hassanie (SCIPER 404885) Cristina Morad (SCIPER 405241)

Submitted: May 8, 2025

Abstract

In this lab, we integrate and evaluate a fuzzing harness for the `tmux` terminal multiplexer using Google's OSS-Fuzz infrastructure. We first establish a baseline by measuring line coverage of the existing `tmux_fuzzer` both with and without the provided seed corpus, observing a significant coverage boost when seeds are present.

We then identify two major code regions not exercised by the baseline harness, implement targeted harness improvements to cover those regions, and finally triage a crash uncovered in the session-detach logic, proposing a patch and assessing its exploitability.

1 Introduction

Fuzzing is a proven technique for uncovering memory-safety and logic bugs in C/C++ code. In this assignment we chose `tmux`, an open-source terminal multiplexer for Unix-like systems, as our target because it (1) has a relatively small OSS-Fuzz integration (only one harness), (2) shows very low runtime coverage, and (3) is widely used daily by developers and DevOps engineers across countless systems. By improving its fuzzing harness, we aim both to increase `tmux`'s coverage under OSS-Fuzz and to demonstrate how targeted harness modifications can uncover real bugs in a piece of critical infrastructure.

2 Methodology

Experiments were conducted on Ubuntu 22.04 LTS with Docker 20.10 and Python 3.10. We used a local fork of OSS-Fuzz in `forks/oss-fuzz` and the `tmux` seed corpus in `forks/tmux-fuzzing-corpus`. Two bash scripts (`run_w_corpus.sh` and `run_wo_corpus.sh`) automate each 4 h campaign. The main steps are as follows:

Reset build and cleanup: the script will first reset the build scripts to the latest version using the `git reset --hard` command. This ensures that the user is working with a clean slate (without any modifications from previous runs). Furthermore, all files in the `build/out` directory are removed to ensure that the user is working with a clean build environment.

Apply patch: the user is running the fuzzing campaign without the seed corpus, the script will apply a git patch named `remove_seed_corpus.patch`. This patch will remove the seed corpus from the build scripts, ensuring that the fuzzing campaign runs without any pre-existing input seeds.

Build with coverage: in order to build the fuzzing harness with coverage instrumentation, we use the `build_image` and `build_fuzzers` commands from the `helper.py` script.

```
cd forks/oss-fuzz
python3 infra/helper.py build_image \
  tmux --pull
python3 infra/helper.py build_fuzzers \
  --sanitizer coverage tmux
```

The `-sanitizer coverage` flag ensures that

the fuzzing harness is built with coverage instrumentation, allowing us to measure the code coverage during the fuzzing campaign.

Prepare corpus:

- Seeded run: copy all files from `forks/tmux-fuzzing-corpus` into `build/out/corpus`.
- Unseeded run: ensure `build/out/corpus` is empty.

Run fuzzing campaign: the script will run the fuzzing campaign using the `run_fuzzer` command from the `helper.py` script. The `--max_total_time` flag is set to 14400 seconds (4 hours) to limit the duration of the fuzzing campaign.

```
python3 infra/helper.py run_fuzzer \
  --engine libfuzzer tmux \
  --corpus-dir build/out/corpus \
  input-fuzzer \
  -max_total_time=14400 \
  -timeout=25 \
  -print_final_stats=1 \
  -artifact_prefix=./crashes \
  -jobs=$(nproc) \
  -workers=0
```

Generate coverage: finally, the script will generate the coverage report using the `coverage` command from the `helper.py` script, and copy the coverage report to the `assignment/part_1` directory.

```
python3 infra/helper.py coverage \
  tmux \
  --corpus-dir build/out/corpus \
  --fuzz-target input-fuzzer
```

The `--corpus-dir` flag is set to `build/out/corpus` to ensure that the coverage report is generated using the same corpus used during the fuzzing campaign.

3 Part 1: Baseline Evaluation

3.1 With Seed Corpus

List the exact build/run commands and point to `run_w_corpus.sh`.

3.2 Without Seed Corpus

List the exact build/run commands and point to `run_wo_corpus.sh`.

3.3 Coverage Comparison

Discuss coverage percentages and key observations.

4 Part 2: Coverage Gaps

4.1 Region A

Justification of significance; why existing harness misses it.

4.2 Region B

Justification of significance; why existing harness misses it.

5 Part 3: Fuzzer Improvements

5.1 Improvement 1 (Region A)

Describe changes, refer to `improve1/run_improve1.sh`, and summarize coverage delta.

5.2 Improvement 2 (Region B)

Describe changes, refer to `improve2/run_improve2.sh`, and summarize coverage delta.

6 Part 4: Crash Analysis

Detail crash reproduction (`run_poc.sh`), ASAN log snippet, root cause, proposed patch, and exploitability.

7 Conclusion and Future Work

Summarize achievements and outline possible next steps.