Lab 4 - Fuzzing

> Secure System Development - Spring 2025

In this lab, we will practice WebApp and binary fuzzing.

- Create a `.md` step-by-step report of the actions you took with screenshots of key results.

Task 1 - WebApp Fuzzing

> Guide: https://github.com/ffuf/ffuf#example-usage

- Install ffuf or wfuzz and SecLists
- Run DVWA locally with docker

  ```
  docker run -d -p 127.0.0.1:80:80 vulnerables/web-dvwa
  ```

- Fuzz for endpoints using the appropriate wordlists and flags from SecLists. Example with `fuff`

  ```
  ffuf -u http://localhost:80/FUZZ -w <WORDLIST>
  ```

- Answer the questions by fuzzing using the wordlists at `<SECLISTS_DIR>/Discovery/Web-Content` and filtering results.
  - Which endpoints/files from `big.txt` were accessible? Which ones gave interesting error codes (not 404).
  - What file extensions from `web-extensions.txt` are available for the `index` page?
  - Which directories from `raft-medium-directories.txt` are accessible? Which ones gave interesting error codes (not 404).
- Include in your report
  - Command used to answer a question
  - Text explanation of what the command does
  - Screenshot from YOUR terminal showing the command result

Task 2 - Python Fuzzing

> Guide: https://afl-1.readthedocs.io/en/latest/user_guide.html

Given this intentionally-buggy Python code that takes user input and does URI decoding. Use the python-afl to fuzz-test the program [see instructions below].

```
import afl
import sys

def uridecode(s):
    ret = []
    i = 0
    while i < len(s):
        # Translate %xx to its corresponding ASCII character
```

```
        if s[i] == '%':
            a = s[i + 1]
            b = s[i + 2]
            char_code = (int(a, 16) * 16) + int(b, 16)
            ret.append(chr(char_code))
            i += 3

        # Translates '+' into space
        elif s[i] == '+':
            ret.append(' ')

        # Leave other characters unchanged
        else:
            ret.append(s[i])
            i += 1
    return ''.join(ret)

if __name__ == '__main__':
    afl.init()
    print(uridecode(sys.stdin.read()))
```

Instructions

1. Install AFL++ locally (e.g., with `apt`), or run it in docker with the following commands:

```
# Run AFL++ in docker, mounting cwd as a volume
docker run --name afl -ti -v .:/src/ aflplusplus/aflplusplus

# Install python-afl
pip install python-afl
```

2. Prepare `input` directory with appropriate test input (i.e., seed corpus).
3. Run the fuzzer with

```
py-afl-fuzz -i input -o output -- /usr/bin/python3 main.py
```

4. Wait for a while, expect at least one detected crash and one detected hang.
5. Analyze the results:
   - Show `fuzzer_stats` and some input that caused crashes/hangs.
   - Reproduce a program `crash` and a program `hang` case using the obtained results.
   - Explain the problems and why they happened, then propose a fix.
6. Give **brief** and **concise** answers to these questions (in your **own** words)
   - Will the fuzzer ever terminate in the above experiment? Why/Why not?
   - How coverage-guided fuzzers work? Is AFL coverage-guided?
   - How to optimize a fuzzing campaign?