CISPA
HELMHOLTZ CENTER FOR
INFORMATION SECURITY

Security Testing

WS 2023/2024

Prof. Dr. Andreas Zeller
Leon Bettscheider
José Antonio Zamudio Amaya

Exercise 1 (10 Points)

Due: 12 November 2023 23:59

The lecture is based on The Fuzzing Book, an *interactive textbook that allows you to try out code right in your web browser*.
The Fuzzing Book code is additionally available as a Python pip package. To work on the exercises, please install the package locally:

```
pip3 install fuzzingbook
```

Submit your solutions as a ZIP file on your status page in the CMS.

We will provide you a structure to submit your solutions where each task has a dedicated file. You can add new files and scripts if you want, but you may not delete any provided ones. You can verify whether your submission is valid by executing `verify.py`:

```
python3 verify.py
```

The output provides an overview if a required file, variable, or function is missing and if a function pattern was altered. If you do not follow this structure or change it, we cannot evaluate your submission. A non evaluable exercise will result in 0 points, so make sure to verify your work before submitting it. Note that the script does not reveal if your solutions are correct.

Exercise 1-1: Ohh Look, a Bug (4 Points)

In this exercise you need to find bugs in a faulty implementation of the Levenshtein distance, which you can find in **exercise_1.py**.

a. I found a Crash (1 Point)

The implementation of the distance produces in some cases an `IndexError`. Find the conditions under which this error is triggered and provide values for the arguments `s1` and `s2` in **exercise_1a.py**.

b. The Strange Thing (1 Point)

Beside of the `IndexError` there is still something wrong with this implementation. However, this bug does not produce an error. Provide values for the arguments `s1` and `s2` in **exercise_1b.py** that are suited to reveal this bug, i.e. such that the result is altered by it.

Make yourself familiar with the concept that a fuzzer can indeed trigger the error from exercise 1-1 a. but will not reveal the bug from this exercise unless it propagates and produces a crash, because a fuzzer does not verify the logic of a program.

c. Fix Me (2 Points)

Correct both issues of the function and implement your solution in **exercise_1c.py**.

Exercise 1-2: Fuzz it till you Crash it (6 Points)

In this exercise you should trigger the bug in the Levenshtein distance by applying the concepts from the chapter Fuzzing: Breaking Things with Random Inputs. Implement all your solutions for this exercise in **exercise_2.py**.

a. Put the Fun in Functions (3 Points)

Extend the `ProgramRunner` Class such that it can execute a function. You can follow this guideline to accomplish this:

- Override the `run_process()` method so that it can execute a given function. The function is given by the field `self.program`. In Python, a function can be stored in variables or could be given as an argument, for example in the follwing code. `sum()` is a function that adds all elements of an iterable. `f()` takes the function `g()` and applies it to the argument `x`.

In [8]:

```
def f(g, x):
    return g(x)

f(sum, [1, 2, 3])
```

Out[8]: 6

- Override the `run()` method. The method should return as the result a tuple of the `inp` argument and the return of the function provided in `self.program` (you should use the `run_process` method for this) if it exists and `None` otherwise. As the outcome the method should return `self.FAIL` if the input triggers an exception of the type `LookupError` in the function `self.program`. For any other exception or error the outcome should be `self.UNRESOLVED`. If no exception or error is triggered, the outcome should be `self.PASS`. To accomplish this you could make yourself familiar with Try and Except in Python.

b. Wrap it (2 Points)

Build a wrapper for the function `levenshtein_distance()` from exercise 1-1. This is required because in our current setup our `Fuzzer` and `Runner` classes can only process one input at a time. To overcome this issue you should implement the `ld_wrapper` function. The function should split the input whenever the character `'+'` occurs and should use the first two possible splits as the arguments for the `levenshtein_distance`. Consider the string `'a+b+c'`, it should be split in `'a'`, `'b'`, and `'c'` and the `levenshtein_distance` should then be called with `'a'` and `'b'`. If fewer than two splits exist, i.e. if the string does not contain a single `'+'`, you should raise a `ValueError`. Do not forget to return the result of the `levenshtein_distance`.

c. A (not so) Random Fuzzer (1 Point)

For this part you should configure and setup the `RandomFuzzer`, such that it usually should find the error in the `levenshtein_distance()` function within *10* trials. To do so, please modify the construction of the `RandomFuzzer` inside the `run()` function by providing arguments for it as specified in the Fuzzing Book.