CISPA HELMHOLTZ CENTER FOR INFORMATION SECURITY

Security Testing

WS 2023/2024

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

Exercise 11 (10 Points)

Due: 28. January 2024

> 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 11-1: Compiler Fuzzing with ISLa Constraints (8 Points)

In this exercise you will use ISLa constraints to customize the PythonFuzzer introduced in the FuzzingBook chapter on Compiler Fuzzing (https://www.fuzzingbook.org/html/PythonFuzzer.html). It is recommended to review the FuzzingBook chapter on Fuzzing with Constraints (https://www.fuzzingbook.org/html/FuzzingWithConstraints.html) and the ISLa language specification (https://isla.readthedocs.io/en/latest/islaspec.html).

In each subexercise (1a, 1b, 1c, 1d) you should write an ISLa constraint that satisfies the stated requirement.

You may only modify the variable `constraint` in exercise_1a.py, exercise_1b.py, exercise_1c.py exercise_1d.py. The type of `constraint` must be `str`. The file exercise_1.py must not be changed.

For example, if you want to test the constraint you have written for exercise 1b, you can invoke exercise_1.py as follows:

`python3 exercise_1.py 1b`

Note that there is a **hard time limit of 5 minutes** for each subexercise. That is, your solution is only valid if it generates the intended program within 5 minutes.

a. First constraint (2 Points)

Requirement: *Each function definition must contain exactly three break statements.*

b. Second constraint (2 Points)

Requirement: *The program must contain at least one while loop, which contains at least one break statement in its body.*

c. Third constraint (2 Points)

Requirement: *The program must contain at least two if statements, which are not nested.*

d. Fourth constraint (2 Points)

Requirement: *The program must contain at least one return statement with a return expression that contains an integer constant with a value greater than 995 and smaller than 1005.*

Exercise 11-2: Quiz (2 Points)

In this exercise we will recap the chapter on *Compiler Fuzzing*.

Provide the BEST answers to the following questions in `exercise_2.py` by assigning to each variable `Q1, Q2` the values `1` to `4`.

For instance, if you think the first answer is the BEST answer to Q1, set `Q1=1` in `exercise_2.py`.

There is only **one BEST answer** to each question.

Questions

Q1: Why is ISLa useful for testing compilers?

1. It's based on SMT solving, which allows it to generate much more inputs per second than regular grammar fuzzers.
2. It's tailored specifically towards testing the Python programming language.
3. Semantic constraints can be specified in ISLa, which allows to focus the fuzzer on very specific parts of the compiler.
4. It supports differential greybox fuzzing out of the box.

Q2: What is the main benefit of using an AST grammar as shown in the Compiler Fuzzing chapter?

1. The Python AST module already provides ISLa constraints.
2. We can generate more inputs this way than writing the grammar completely by hand.
3. Floats can be treated just like integers without any further changes.
4. We can use existing Python parsing infrastructure and do not have to implement details such as whitespaces.