CISPA HELMHOLTZ CENTER FOR INFORMATION SECURITY

Automated Debugging

WS 2022/2023

Prof. Dr. Andreas Zeller
Paul Zhu
Marius Smytzek

Exercise 0 (0 Points)

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 must not delete any provided ones. You can verify whether your submission is valid by calling python3.

```
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.

This is the usual header for each exercise, but you do not need to submit this exercise 0.

Exercise 0-1: Snake City (0 Points)

In this exercise you will install Python and run it.

a. Above C Level (0 Points)

Install Python 3.10 by following the steps on https://wiki.python.org/moin/BeginnersGuide/Download. Please install Python 3.10 because other versions of Python may not work with the dependencies of this lecture.

b. Do Pythons run? (0 Point)

Run the python script **exercise_1b.py**. Verify that the script prints `The Debugging Book`.

c. The Framework (0 Point)

We will provide a simple framework for your submissions. For each assignment we provide a **verify.py** file that validates whether all files, functions, and variables that are required for the assignment exist in your project. We ask you to verify your submissions before handing in, because we can only grade valid submissions. For this exercise, please execute **verify.py** and check its output.

Exercise 0-2: Zeus, or Something more Roman? (0 Points)

a. My Binder (0 Points)

You can directly edit, experiment, and try the code from The Debugging Book on a mybinder server. To do so, please navigate to the chapter about Introduction to Debugging, hover over Resources, and click on Edit as Notebook. Try out the entire code in the notebook. You can follow these steps with any other chapter of The Fuzzing Book. Do not despair, the server requires some time to start (≈30 seconds).

b. Jupyter (0 Points)

Install Jupyter Notebooks by following the steps on https://jupyter.org/install . We would recommend to install JupyterLab via pip, i.e., pip3 install jupyterlab.

c. Notebooks (0 Points)

Start a Jupyter notebook server on your machine by running jupyter notebook from the directory of this exercise sheet. Your browser should automatically open the index page of the server, if not open the link provided by jupyter notebook. Open the notebook **exercise_2b.ipynb** in the shown environment and follow the next steps.

1. Create a new markdown cell with an arbitrary content.
2. Execute the two code cells.
3. Delete the markdown cell with the content Delete this cell.

Do not forget to save your changes.

Exercise 0-3: The Debugging Book (0 Point)

Install the Python package from The Fuzzing Book by running

```
pip3 install debuggingbook
```

This package allows you to access each function and class from The Debugging Book's code from within your Python programs by importing it.

Exercise 0-4: The Pythianionic Way (0 Points)

In this exercise you will try some of Python's capibilities to produce elegant, simple, and understandable code. You should try out and experiment with Python to find the correct solutions. You can try your implementations in **exercise_4.py**. You can run the script to verify your results.

a. Shorter (0 Points)

Take a look at the average function below. In Python, the body of this function could be expressed in a single statement. Find another implementation of this function that comprises only one statement.

In [5]:
```python
def average(l: list) -> float:
    """
    Calculates the average of a number list.
    """
    s = 0
    for i in range(len(l)):
        s += l[i]
    return s / len(l)
```

b. Is This Even Possible? (0 Points)

Now take a look at the middle function below. The goal of this exercise is to remove all `and` operators from the code while keeping the structure of this program, i.e., the three If-statements should preserve their semantics.

In [10]:
```python
def middle(x: int, y: int, z: int) -> int:
    """
    Finds the middle of three arguments x, y, and z
    that is neither the maximum nor the minimum.
    """
    if (x <= y and y <= z) or (z <= y and y <= x):
        return y
    elif (y <= x and x <= z) or (z <= x and x <= y):
        return x
    else:
        return z
```

c. Fun with Lists (0 Points)

Implement the following functions:

In [15]:
```python
def reverse_list(l: list) -> list:
    """
```

```
    Returns l in reversed order,
    e.g., reverse_list([1, 2, 3]) == [3, 2, 1]
    """
    pass

def sort_list(l: list) -> list:
    """
    Returns l in ascending order,
    e.g., sort_list([3, 1, 2]) == [1, 2, 3]
    """
    pass

def each_other(l: list) -> list:
    """
    Returns every second element of l starting from the first,
    e.g., reverse_list([1, 2, 3, 4, 5]) == [1, 3, 5]
    """
    pass

def all_even(l: list) -> list:
    """
    Returns all even elements of l in order,
    e.g., reverse_list([4, 3, 2]) == [4, 2]
    """
    pass
```