CISPA
HELMHOLTZ CENTER FOR
INFORMATION SECURITY

Security Testing

WS 2021/2022

Prof. Dr. Andreas Zeller
Leon Bettscheider
Marius Smytzek

Exercise 2 (10 Points)

The lecture is based on The Fuzzing Book (https://fuzzingbook.org/beta), 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:

```
pip install --extra-index-url https://test.pypi.org/simple/ fuzzingbook==1.0rc2
```

Submit your solutions as a `.zip` file on your status page in the CMS (https://cms.cispa.saarland/fuzzing2122/students/view).

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 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.

In this exercise sheet, you will write context-free grammars for two programming languages.

Exercise 2-1: (Brainf*ck) (5 Points)

Please familiarize yourself with the brainf*ck programming languages by reading the wikipedia page (https://en.wikipedia.org/wiki/Brainfuck).
In particular, focus on the eight commands of the programming language: `> < + - . , [ ]`

a. Write a grammar (2 Points)

Please write a context-free grammar in fuzzingbook format (https://www.fuzzingbook.org/html/Grammars.html) for the brainf*ck programming language.
The start symbol should be `<start>`. This grammar should be able to produce the set of all brainf*ck programs.
Make sure that the programs your grammar produces do have balanced parentheses.
Store the grammar in **bf_grammar.py**. The grammar's variable name should be `BFGRAMMAR`.
Before submitting, please make sure your grammar is valid by running `assert is_valid_grammar(BFGRAMMAR)`, or, alternatively run the `verify.py` program which also performs this check.

b. Just fuzz it (3 Points)

We implemented a brainf*ck interpreter in **bf.py**. Unfortunately, we felt very dizzy during programming and introduced **4** bugs (1 bug per line) to the program.
Please use **fuzzBF.py** with the grammar you've written in `Exercise 2-1a` to find those bugs using fuzzing. Logs are written to stdout and help you to diagnose the errors.
Running the fuzzer **1000** times is sufficient to solve this exercise.
Please write your solutions for this exercise to **exercise_1b.txt**.

b-1. Where do errors become apparent? (1 Point)

List all the lines in **bf.py** where an error occured during fuzzing. Note that the line where an error occured is not necessarily the line where it originated from (root cause).

Note: `timeout` does not indicate a bug. Timeouts will happen as the grammar can generate programs that are not guaranteed to terminate.

b-2. Where do the errors originate? (2 Points)

All the errors that can be observed during fuzzing originate from **4** faulty lines of code.
Please give the line numbers of the **4** faulty lines in **bf.py** and explain the fault for each line (one sentence each).

Exercise 2-2: TinyC (3 Points)

TinyC is a simplified subset of the C programming language.
The TinyC language is characterized by the following context-free grammar in BNF notation (http://www.iro.umontreal.ca/~felipe/IFT2030-Automne2002/Complements/tinyc.c):

```
<program> ::= <statement>
<statement> ::= "if" <paren_expr> <statement> |
                "if" <paren_expr> <statement> "else" <statement> |
                "while" <paren_expr> <statement> |
                "do" <statement> "while" <paren_expr> ";" |
                "{" { <statement>* } "}" |
                <expr> ";" |
                ";"
<paren_expr> ::= "(" <expr> ")"
<expr> ::= <test> | <id> "=" <expr>
<test> ::= <sum> | <sum> "<" <sum>
<sum> ::= <term> | <sum> "+" <term> | <sum> "-" <term>
<term> ::= <id> | <int> | <paren_expr>
<id> ::= "a" | "b" | "c" | "d" | ... | "z"
<int> ::= <an_unsigned_decimal_integer>
```

Translate the grammar given above to fuzzingbook syntax. The start symbol should be `<start>`.
Store the grammar in **tinyc_grammar.py**. The grammar's variable name should be `TINYCGRAMMAR`.
Before submitting, please make sure your grammar is valid by running `assert is_valid_grammar(TINYCGRAMMAR)`, or, alternatively run the `verify.py` program which also performs this check.

Optional: To see your grammar in action, first compile the TinyC compiler (**tinyc.c**) using your favorite C compiler, e.g. `gcc tinyc.c -o tinyc`. The resulting executable should be named **tinyc**. Next, run **fuzzTinyC.py** and marvel at the programs your grammar generates.

Exercise 2-3: Limitations of context-free grammars (2 Points)

As we have seen, context-free grammars can be used to describe the set of Brainf*ck and TinyC programs.
Please give two arguments (1-2 sentences each) that explain why we **cannot** describe arbitrary programming languages precisely using context-free grammars.
Store your answer in **exercise_3.txt**.