# P3: Fuzzing with AFL

Jaco van de Pol, Freark van der Berg, Maarten Everts

**Rules of the game:**

1. Please work in groups of 2 persons, following the Blackboard groups.
2. The work should be submitted on Blackboard under "Assignments".
3. The "core" part (A) is obligatory. (B) and/or (C) are "bonus parts".
4. The submission consists of 2/3/4 files: exactly one "pdf" (report on A and B or C), and at least a "zip" (all code and results for part A). Optionally, you may provide separate zips for part B or part C. (Each of them is good for a bonus)
5. Clearly indicate who participated in which part of the assignment.
6. The "report" should contain:
   - Your names and s-numbers, and roles in the various parts
   - Screenshot(s) of the afl-window of the final results after fuzzing is terminated.
   - A short description of your findings (max 1 A4 text for parts A and B, max 2 A4 text for part C)
   - A short statement how to reproduce your experiment (tools, versions, parameters, etc.)
7. The deadline for this final assignment is: **January 10, 23:59 CET**.
8. Submissions by E-mail instead of Blackboard, after the final deadline, or in the wrong zip+pdf format will be ignored.

# A: Fuzz your own Sorted Tree Implementation

The goal is to use fuzzing with AFL to find out if there are still some vulnerabilities remaining in your C-implementation of the Sorted Tree implementation (SINT) from Assignment (A).

You can proceed as follows (in reality you probably need a couple of iterations to get useful results):

1. Install AFL (the American Fuzzing Lop) on your computer (`http://lcamtuf.coredump.cx/afl/`) and read the documentation on the website and included in the distribution.
2. Adapt the Makefile for your SINT-code, to instrument it with AFL (using e.g. afl-gcc or afl-clang, depending on your platform).
3. Provide a small input file with SINT-commands in some input directory
4. Start fuzzing with afl-fuzz. You have to terminate it manually at some point. Capture a screenshot of the output upon termination.
5. Inspect the reported crashes (input files) if any, and try to reproduce them (there might be spurious alarms). Report the number of real and spurious crashes.
6. Debug your code, based on crashing inputs (if any), preferrably using a debugger like gdb.
7. Repair your program until you cannot find new errors. Also list the repairs in the document.

Optional Try if enabling ASAN (address sanitizer), or even using Valgrind with AFL, uncovers more errors

8. The "zip" for part A should contain:
   - A directory with the original code that you are fuzzing, including scripts to reproduce building and fuzzing the code.
   - The `input` and `output` directories used for fuzzing
   - A directory with your improved, hopefully fault-free and secure, code, including the *original* Makefile. (follow the guidelines of Project 1, so we can test it automatically).

## B: fuzzing Rust programs and libraries (bonus)

Fuzz testing can also be applied to Rust source code. Even though the problems found this way typically do not result in in dangerous buffer overflow and other memory-safety-related issues, it can help uncover unexpected crashes or logic problems. See `https://github.com/rust-fuzz/afl.rs` for documentation on how to apply AFL to Rust source code.

Try to apply AFL to (parts of) your text-statistics program you created in the previous assignment and report on the errors or unexpected panics AFL can find. If needed and doable, try to make your program more robust based on AFLs findings.

Alternatively, try to find an interesting "fresh"[1] Rust library and apply AFL to it and report on your findings. Since Rust is a fairly young language, there are many "fresh" libraries available for testing.

## C: writing secure code (bonus)

Select a **fresh**[2] open source project in C (or C++) and try to fuzz it. Remember that fuzzing makes sense on applications that require some (structured) input from stdin or from some file.

In the report, describe clearly the origin of the code that you analyzed, and how you conducted your experiment. Include a screenshot of the final window from AFL, and shortly describe your findings, e.g. the found bugs (if any).

In a separate zip, include the code that you fuzzed, the script/makefiles to rerun the fuzzing experiment, and the input- and output-directories for fuzzing.

[1] Do check whether the library/program is not yet listed on `https://github.com/rust-fuzz/trophy-case`

[2] this excludes packages for which a fuzzing report/status exists on the web