Assignment 3 — Secure Programming 2020

Part 1 — 10%

Question 1 — 10%

Explain how you solved the puzzle that gives access to this assignment.

Part 2 — 90%

In this assignment you will fuzz a memory allocation library. You should write a *driver* program that exercises the library based on the input it accepts. The program should support one of two behaviours:

- **Pseudo-random driver:** The program reads a number from `stdin`, uses this number to initialise a pseudo-random number generator, which it uses to determine the sequence of operations on the heap. For this, we provide two functions:
  - `void sp_seed(uint32_t seed)`, which takes a seed and initialises a pseudo-random-number generator.
  - `uint32_t sp_rand()`, which returns the next pseudo-random number.

  To ensure reproducibility, you may only use our random number generator as a source of randomness.
- **A driver for an external fuzzer:** The program assumes that an external fuzzer, such as AFL, produces a sequence of characters, which the program reads from `stdin`. This sequence of characters determines the sequence of operations the program performs.

Your driver program should be deterministic, i.e. if it gets the same input, it should performs the same sequence of operations. The sequence of operations it generates should be correct, and should never fail if the underlying malloc library is correct.

malloc library:

The file `spalloc.c` contains an implementation of a malloc library. To facilitate debugging, we renamed the functions to `sp_malloc`, `sp_calloc`, `sp_free`, and `sp_realloc`. However, with the exception of the bugs we introduced, the functionality is identical to `malloc`, `calloc`, `free`, and `realloc`, respectively. The code is only tested on 64-bit Linux. It is unlikely to work on 32-bit Linux. We do not know what happens on Mac or on Windows. We also provide the matching header file `spalloc.h`.

To further assist with debugging, compiling `spalloc.c` with the flag `-DSP_FAKE` replaces the implementation with calls to the standard `malloc` library, providing a known correct implementation. Finally, compiling with `-DSP_ASSIGNMENT1` builds code suitable for use as a solution to Assignment 1. (`SP2020_HeapStats()` is not functional and is only provided for compatibility with the soundness test of Assignment 1.)

To build the code, you need to modify the code, by setting the value of the macro `STUDENTNUM` to your student number. (Seven digits, without the 'a'.)

The code contains three known non-trivial bugs. Your aim is to identify, triage and fix these bugs. To get marks for any of the bugs, your submitted solution should include an input that causes your test program to crash. Note that changing `STUDENTNUM` may change the behaviour of the code. To get marks, the fail

should happen when `STUDENTNUM` is set to your student number. You will not get any marks for describing or correcting bugs that you cannot reproduce with your student number and your driver program.

Marking

1. Driver software – 30%. Awarded based on both the code and the description of how the driver is built.
2. First bug – 30%.
3. Second bug – 20%.
4. Third bug – 10%.
5. Any additional bug – 5% bonus. There is no bonus for trivial bugs, for bugs in `SP2020_HeapStats()`, for bugs related to running out of memory, for bugs that cannot be reproduced on 64 bit Linux, or for any bug that only appears when compiling `spalloc.c` with non-default flags (including the defines mentioned above).

Note that you need to demonstrate reproducible bugs. Hence, when you find an input that causes a bug **save a copy of the software and inputs** used. See the submission instructions for requirements.

Submission Instructions

You should submit a `.tar` or a `.tgz` archive. The archive should contain a single directory, whose name is your student a-number. In that directory, we expect to find the following:

- A text file named `info.txt`, which contains two lines. The first line is your name, and the second is your student number.
- A PDF document named `assignment3.pdf`, which contains the answer to Parts 1 and 2. For Part 2 the answer should contain a description of how your driver software works and what it tests for, and a description of the bugs you have identified, and how you fixed them. If you did not fix some of the bugs, you need to justify why you claim these are not the same.
  The file should be typed-up, i.e. scanning a handwritten paper is not acceptable. Also, the file must be a PDF document. Text or Word documents are not acceptable, even if you changed the extension to `.pdf`.
- A file named `driver.c`, which contains the code to your driver. To build, we will use the command `cc -o driver driver.c spalloc.o`, where `spalloc.o` is the object code generated from `spalloc.c`.
- Optionally, a file named `spalloc-fixed.c`, containing `spalloc.c` with fixes to all of the bugs you identified, triaged, and fixed.
- Optionally, sub-directories names `bug1`, `bug2`, `bug3`... Each of these subdirectories should contain an input file named `input.txt`, which reproduces the bug when provided to you driver program.
  Because your driver program may change during the work and to allow incremental discovery of bugs, each of the folder may optionally include a version of `driver.c` and of `spalloc.c`. If any of these exist, we will use these for reproducing the bug. Otherwise, we will use your submitted `driver.c` and the original `spalloc.c` as required. Note that the `spalloc.c` files included in bugs folder may only contain fixes for earlier bugs. That is, the code in `bug2` may include a fix to the bug reported in `bug1`, but no other modifications.

We will provide a soundness test script for this assignment. By now the use of the script and the implications of not passing it should be well understood.