Assignment 2 — Secure Programming 2020

Part 1 — 30%

Question 1 — 10%

Download the program `offset.c` from the assignment description page and build it using:

```
cc -O3 -o offset offset.c
```

The program takes a single argument, also called `offset`, does some computation, and output the time it took for the computation to run. Try the program with several offsets. In particular, note the timing difference between offsets 16 and 17. (You may want to run each offset multiple times to get the potential range of results.)

Explain the difference and how it relates to Part 2 of Assignment 1.

Question 2 — 10%

Download the program `skip.c` from the assignment description page and build it using:

```
cc -O3 -o skip skip.c
```

The program takes a single argument, also called `skip`, does some computation, and output the time it took for the computation to run. Try the program with several skips. In particular, note that skip 512 shows a different timing from that of other skips in the range 500–520. (Again, you may want to run each skip multiple times to get the potential range of results.)

Explain the cause of the behaviour of skip 512. You may want to refresh your memory about set-associative caches.

Question 3 — 10%

Both programs investigated in this part end with the `if` statement

```
if (s != 0)
    printtime(end - start);
```

Why is this statement required?

Part 2 — 70%

In this part of the assignment, you will extend the memory allocation you implemented in Assignment 1 to also checks for potential misuses. Below we list the events you need to identify and abort the execution of the program if they occur. One option is to configure the system to cause a fault when the event occurs. As an example, if you unmap large chunks, accesses after `free` will cause a segmentation violation. Alternatively, when you identify the event, you can manually abort the program by calling `abort()`.

Double Free: A double free event occurs when a program frees a buffer it has already freed. You only need to identify such an event if the second call to `free()` occurs before the next time the same memory is allocated to the program.

Use After Free: Use after free is a software bug in which the program accesses a location it had earlier freed. Your library should detect a write that occurs between a free and the following allocation of the same

memory location. Detection should happen at the latest when the location is allocated again. As with all detection of writes, you do not need to detect writes that do not change the value. To protect from read after free, your program should either ensure that the contents of the chunk is scrubbed after free, or that an exception occurs at read.

Invalid free: Detect a call to `free` with a non-NULL pointer that was not returned from `malloc`, `calloc`, or `realloc`.

Buffer Overflow/Underflow: Your library should detect an overflow or an underflow that changes any of the 8 bytes directly after or directly before the allocated buffer. You should make effort to detect changes at larger distances, as long as detection does not require allocation of more memory. Writes should be detected at the latest when the affected chunk is freed or reallocated.

Heap Test: You should also implement the function `SP2020_HeapTest()`, which ensures that tests the whole heap to identify if any of the events above is yet to be detected. In particular, the heap test should detect all overwrites and underwrites of up to 8 bytes, for which the affected chunk has not been freed or reallocated. It should also detect all writes to freed memory where the chunk has not been allocated again.

Problems and Limitations

Probabilistic Detection: In some cases it may be infeasible to guarantee that an event is detected correctly. For example, if your library preallocates all chunks of some size, and the program happens to call `free` with a pointer to a buffer in one of the preallocated chunks, the library may detect this as a double free instead of an invalid free. Depending on the library implementation, there may also be cases that events will not be detected. In all such cases you may resort to methods that are likely to detect events, but may fail in rare circumstances.

Question 4 — 35%

Describe how your library detects or prevents double free, use after free, and invalid frees. The description should explain when the event is detected. It should also clearly identify limitations of the method, for example, the scenarios under which the library will fail to detect the event.

Question 5 — 35%

Describe how your library detects buffer overflow/underflow. The description should explain when the event is detected. It should also clearly identify limitations of the method, for example, the scenarios under which the library will fail to detect the event.

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 four items:

- 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 `assignment2.pdf`, which contains the documentation you are asked for. 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 folder named `part2` with the implementation of your memory allocation library. Typing `make` in this folder should build `libmalloc.so`. The expected use of `libmalloc.so` is through the environment variable `LD_PRELOAD`. For example, the command `LD_PRELOAD=./libmalloc.so ls` runs the program `ls` with your memory allocation library. Note that we plan to look at your code, so please make sure it is readable. Also, we will check it with adversarial input and expect it to be at least somewhat robust.

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.