Chapter 4

Simulation

This chapter presents the details of our implementation of Waffle.
We implement and evaluate the performance of Waffle on an Ubuntu 18.4.5 LTS operating system. The computer consists of 16GB of RAM and Intel® Core™ i7-3770 CPU.

4.1 Implementation

We have implemented Waffle on Memlock which is based on AFL fuzzer. To develop Memlock, we added and modified approximately 400 lines of code in C programming language. We also implemented the instrumentations under LLVM framework.

4.2 Instrumentation

To inject code into the binary of the target program, we modify the two files *waffle-llvm-rt.o.c* and *waffle-llvm-pass.so.cc*. The first file is responsible for the initial setup for the SHM and forkserver and etc. and the later file injects the basic-block level instrumentation which contains our feature collection procedure.
In the file *waffle-llvm-rt.o.c* first we specify an array of 64KB, in addition to the

shared memory implemented in Memlock. This array collects the instruction counters and monitors the execution of the program through each basic-block.

```
    u32  __wafl_icnt_initial[ICNT_SIZE];
    u32* __wafl_icnt_ptr = __wafl_icnt_initial;
```

Listing 4.1: waffle-llvm-rt.o.c

Here $ICNT\_SIZE$ is equal to 2^{14} words, that makes the size of to 64KB.

In the file *waffle-llvm-pass.so.cc* we implement the LLVM pass which helps us with injecting basic-block level instructions. For our purposes, we are using the instruction visitor which we explained in Chapter 2.

After the modifications, we can *make* the LLVM project within Waffle and our changes will be applied in the *waffle-clang*, which we can use for compiling the target program.

4.3 Program inputs and setups

4.4 Generating fuzzed data

4.5 Program execution and monitoring

4.6 Report vulnerabilities