Project 1 : Simple Fuzzer

March 7, 2017

The subjects for the course projects are hosted as a public project on our Gitlab `https://securitytesting.cispa.saarland/kampmann/subjects`. Each subject resides in a top level directory and you can invoke them using:

```
python <subject>/<module> <inputfile>
```

Depending on your setup you might need to substitute `python` for `python3` if your system uses Python 2.x by default. The subjects will terminate with a non-zero exit code in case of an error. Additionally to the public subjects we will also evaluate your implementation on two secret subjects and on variants of the public subjects that contain additional bugs. Your implementation is expected to be accessible as a python module `simplefuzzer.py` in the root directory of your project repository. The fuzzer is supposed to be invokable as:

```
python simplefuzzer.py -s <seed> -t <timeout> -p <path-to-subject-module>
```

The produced failure inducing samples should be written to the working directory using ∗.sample as file ending.

In order to evaluate your implementation we will use 5 seeds provided by you in a text file `seeds.txt` stored in the root directory of your project repository containing one seed per line. Additionally to the seeds provided by you, we will choose 5 additional random seeds. For each seed we will run your implementation with a timeout of 5 minutes (the timeout is given as seconds).

1 Part 1: Fuzzing

Implement a simple fuzzer using the techniques demonstrated in the first lecture. The fuzzer is supposed to be initialized by the provided random seed to make executions deterministic. We consider all inputs that result in a non-zero exit code of the subject and do not raise a ParseException as failure inducing. In order to invoke the subject from your fuzzer, we recommend the use of the subprocess module `https://docs.python.org/3/library/subprocess.html`. Your implementation is expected to write all generated failure inducing inputs into the working directory using `*.sample` as file ending. You are allowed to create temporary directories for generated samples in order to use them as inputs for the provided subject. At the end of the execution these temporary directories and inputs that are not failure inducing are expected to be deleted. The fuzzer will be graded based on the number of unique exceptions that are triggered in each subject.

2 Part 2: Delta Debugging

The second part of this project is the minimization of inputs using delta debugging. Implement and apply the delta debugging algorithm to all failure inducing inputs in order to provide a minimal input that triggers the same exception. The minimization is not invoked as a separate tool. After finding failure inducing inputs during fuzzing your implementation is supposed to minimize them immediately. You are allowed to create temporary directories in the working directory for intermediate inputs that are generated by the delta debugging algorithm but these directories and temporary files have to be deleted at the end of the execution. The minimization will be graded by comparing the size of your minimized inputs to the size of a minimization produced by our reference implementation.