Project 2 : Grammar Based Fuzzer

March 8, 2017

1 Part 1: Fuzzing

Implement a grammar-based fuzzer that, given a context free grammar, is able to produce inputs that are elements of the language defined by the grammar. Make sure to limit the depth of derivation trees to avoid non-termination of the input generation algorithm and exceedingly large inputs. The grammars are given in the format specified by the `grammar` subject. Grammars for the public subjects will be avaliable in the subject folder. The fuzzer must 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 must be deleted. The fuzzer will be graded based on the number of unique exceptions that are triggered in each subject.

2 Part 2: Minimization

The second part of this project is the minimization of inputs using delta debugging. In contrast to the first project the minimization should operate on the derivation trees of the input instead of the resulting character sequence. This will minimize the number of executions of the subjects on subsequences that are syntactically invalid. Implement and apply the delta debugging algorithm to all failure inducing inputs in order to provide a minimal input that triggers the same bug. 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.

3 Part 3: Mutation

We want you to integrate sample inputs and mutations into your grammar based fuzzer. Your implementation must be able to parse provided sample inputs, mutate them and use them in the generation of new samples. The grammars provided with our subjects are all in $LL(k)$ (`https://en.wikipedia.org/wiki/LL_parser`), have no left recursion and are free of ambiguities. In order to parse inputs for a given grammar you can implement a generic greedy recursive descent parser that uses backtracking. Your implementation should parse all sample inputs with the file extension `.sample` in the directory specified with the `-e` parameter and mutate them and use them to generate new inputs.

4 Implementation

The subjects for the course projects are hosted as a public project on our Gitlab `https://securitytesting.cispa.saarland/kampmann/subjects`. Please make sure to pull the most recent revision of the subjects from the project. Each subject resides in a top level directory and you can invoke it 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 `grammarfuzzer.py` in the root directory of your project repository. The fuzzer is supposed to be invokable as:

```
python grammarfuzzer.py -s <seed> -t <timeout> -g <grammar>
                        -p <path-to-subject> -e <examples>
```

The produced failure inducing samples should be written to the working directory using `.sample` as file ending. Like in the first project the timeout is givenin seconds. The grammar is provided as a file and the example parameter points to a directory that contains sample input files ending with `.sample`.

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 .