COM S 413/513: Homework 6 [Project] - Comparing AFL and KLEE

September 29, 2023

Learning Objectives:

In this homework, you will

1. learn the concepts and terminologies related to fuzzing, symbolic execution, bugs, and test input generation
2. gain hands-on experience of setting up and using open-source program analysis tools for bug finding and test input generation

Instructions:

1. Total points: 33 pt
2. Early Deadline: Oct 04 (Wed) 11:59PM
3. Deadline: Oct 06 (Fri) 11:59PM
4. This homework is accomplished by the team of 2 students
5. How to submit:
   - Submit 2 files, a PDF and a Zip.
   - PDF should contain all the answers and screenshots for Q3.
   - Zip should contain all the code and results for Q3 and should have a README to explain which files/folders are associated with each question.

1 Description

In this homework, we are going to run and further study the testing tools learned in our class. American fuzzy lop (AFL) is a fuzzing tool that has found many bugs in real-world software. KLEE is a symbolic execution tool that can automatically generate test inputs for covering as many branches as possible. We will provide a buggy program for you to test and compare the performance of the two tools. In the following, please find a list of steps to follow:

1. Install AFL++, a community maintained fork of AFL.
   - Install the "docker version of AFL++" by following the instructions.
   - Read Quick start: Fuzzing with AFL++
   - Test with the provided example "test-instr.c" found at the base folder in the docker. You might have to run the docker with `--security-opt seccomp=unconfined` to get it to work.

2. Install KLEE:

   - Install the "docker version of KLEE" by following instructions. (also see the video). You might have to run the docker with `--security-opt seccomp=unconfined` to get it to work.
   - Go to `klee_src/examples/get_sign` and run "get_sign.c" example following the tutorial (small correction: under "Replaying test case" use `clang` instead of `gcc`) – sudo access password is `klee`.

3. Prepare the answers for the following:

   (a) (3 pt) Demonstrate the successfulness of installing the AFL by providing screenshots of it running on "test-instr.c". Add the screenshot to the PDF submission.

   (b) (3 pt) Demonstrate the successfulness of installing the KLEE by providing screenshots of it running on "get_sign.c". Add the screenshot to the PDF submission.

   (c) For the function given below:

```c
#include <assert.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
void fuzzMe(int a, int b, char *c) {
  printf("%d,%d,%s", a, b, c);
  if (a >= 20000) {
    if (b >= 10000000) {
      if (b - a < 200) {
        free(c);
        char *s = (char *)malloc(strlen("some random text") + 1);
        strcpy(s, "some random text");
        printf("%s", s);
      } else {
        if (strlen(c) < 7) {
          char *s = (char *)malloc(1);
          strcpy(s, "some random text");
          printf("%s", s);
        }
      }
    }
  }
}
```

      i. (3 pt) Write a test driver/harness in `main()` for AFL. Add a screenshot of the driver function to the PDF submission and add the modified program to the Zip submission.

      ii. (3 pt) Modify this function and write a test driver/harness in `main()` for KLEE. Add a screenshot of the driver function along with the modified function to the PDF submission and add the modified program to the Zip submission.

      iii. (6 pt) How many bug(s) did you find using AFL? Add the screenshot of AFL reporting the bug(s) to the PDF submission and add the output generated by AFL as well as the seed directory to the Zip submission.

      iv. (6 pt) How many bug(s) did you find using KLEE? Add the screenshot of KLEE reporting the bug(s) to the PDF submission and add the output generated by KLEE to the Zip submission.

      v. (9 pt) Reproduce the bug(s) found using AFL using the input generated by the fuzzer and for each "unique" bug(s) document the type of bug and the location in the source code. Add screenshot(s) that shows the program being called with the input(s) as well as the bug(s) being triggered (with the source code location of the error(s)) to the PDF submission.

2 Appendix

Useful commands for docker:

```
# List running containers
docker ps
# List all containers
docker ps -a
# List images
docker images
# Start a pointer and attach a folder <pathInHost> to the container at <pathInHost>
docker run -ti -v <pathInHost>:<pathInDocker> <Image>
# This might be necessary for clang to create executables inside the container
docker run --security-opt seccomp=unconfined -ti -v <pathInHost>:<pathInDocker> <Image>
# Restart a stopped container
docker start <ContainerID>
# Attach to a started container
docker attach <ContainerID>
# Copy a file from host to container
docker cp <fileName> ContainerID:<pathInDocker>/<fileName>
# Copy a file from Docker container to host
docker cp ContainerID:<pathInDocker>/<fileName> <pathInHost>/<fileName>
# Exit docker without kill it
ctrl+p, ctrl+q
```

Useful commands for AFL

```
# Compile with instrumentation
afl-cc -o <output> <input>.c
afl-c++ -o <output> <input>.cpp
# Compile with instrumentation and address sanitization
# (Useful for triggering bugs with location info with fuzzed test input)
afl-cc -fsanitize=address -o <output> <input>
# Fuzz
afl-fuzz -i <seed_dir> -o <output_dir> <fullPathToBinary>
```

Useful commands for KLEE

```
# Compile with instrumentation
clang -I <pathToKleeSource>/include -emit-llvm -c -g -O0 -Xclang -disable-O0-optnone <
    input>.c
# Compile with instrumentation and address sanitization
# (Useful for triggering bugs with location info with fuzzed test input)
clang -fsanitize=address -g <input>.c
# Fuzz
klee <output>.bc
# Fuzz, but only keep new paths
klee --only-output-states-covering-new <output>.bc
# Read the error message and prarameters
cat <output_dir>/<test_id>.err
ktest-tool <output_dir>/<test_id>.ktest
```