CS490: Dynamic Test Generation

Problem statement

In this project, you'll learn how to use KLEE to generate test input.

Install KLEE

DO NOT use ector or lore for this project. Use linux machine such as sslab or data.cs.purdue.edu instead.

1. Download and decompress http://keeda.stanford.edu/~pgbovine/klee-cde-package.v2.tar.bz2.

   ```
   $ curl -O http://keeda.stanford.edu/~pgbovine/klee-cde-package.v2.tar.bz2
   $ tar xjvf klee-cde-package.v2.tar.bz2
   ```

2. Set PATH, environment variable

   ```
   $ export PATH=$PWD/klee-cde-package/bin:$PATH
   $ export PATH=$PWD/klee-cde-package/cde-root/home/pgbovine/llvm-2.7/Release/bin:$PATH
   ```

   Add above lines to .bashrc if you want to do it automatically everytime you login.

How to Use KLEE

Plese read the tutorial first http://klee.llvm.org/Tutorial-1.html.
Note that in the tutorial, the tool named klee is used. However in the package you will be using, all the programs have .cde suffix. Hence, you need to adjust the program name when you follow the tutorial. For example, you should type llvm-gcc.cde instead of llvm-gcc and type klee.cde instead of klee in the tutorial.

In order to test a program, first compile it to its object file then use the following command line. Note that if the source package has multiple source files, we provide a script to compile these files into one object file.

```
$ klee.cde -libc=uclibc -posix-runtime test.o
```

Those two arguments are needed for a program using c library functions.

Assume test.o takes one argument and you want to make it symbolic (meaning all internal values will be computed as equations over that symbolic argument),

```
$ klee.cde -libc=uclibc -posix-runtime test.o -sym-arg 10
```

This will make a one symbolic argument with length up to 10 bytes.
Klee will generate concrete values for the symbolic argument.
You can have multiple symbolic arguments, or a mix of symbolic and concrete arguments. For example, assume test.o has two arguments and we want the second one to be symbolic (i.e. we try to reason about the different behavior of the program when the second input changes), you should use

```
$ klee.cde -libc=uclibc -posix-runtime test.o Argument1 -sym-arg 10
```

You can also make an internal variable symbolic so that that variable is treated as input, using klee_make_symbolic(...) (please refer to the tutorial).

Assume test.o takes one argument which is a file name and it reads data from the file and you want to make the file symbolic,

```
$ klee.cde -libc=uclibc -posix-runtime test.o A -sym-files 1 10
```

-sym-files 1 10 option will generate one symbolic file with size of 10 bytes. Klee will choose file name and we cannot control it. Klee will choose A as a first symbolic file name, B as a second symbolic file name and so on. Hence, we are passing concrete value A to the program as the name of the input file.
The meaning of a symbolic file is that each byte inside the file is treated as a variable that can change. Klee will generate concrete values for each of such variables, which constitute the program input.

Part 1 (9pt)

1. Test the following program with Klee(4pt). p1.c
    a. Modify the source code of the program and make the variable 'a' symbolic using klee_make_symbolic().
    b. Report set of paths covered by the Klee with the symbolic variable.
    c. Select two other input variables and repeat (b) for each variable.
    d. Make all input variables symbolic and find an infeasible path in the program.
    e. Submit set of paths covered by for each of three variables(3pt) and infeasible path(1pt).

2. Test the buggy gnupg with Klee and find a bug(5pt).
    a. Download buggy gnupg from here and build with including script

    ```
    $ curl -O http://www.cs.purdue.edu/homes/kim1051/cs490/proj3/gnupg-buggy.tar.gz
    $ tar xzvf gnupg-buggy.tar.gz
    $ cd gnupg-1.0.5
    $ cd build
    $ ../configure --disable-nls --with-included-zlib --disable-dynload --with-included-gettext
    $ sh make.sh
    ```

    b. Try Klee with one concrete argument "-no-armor" and one symbolic file on gpg.bc built in the previous step.
    c. Report a testcase that causes a bug. Ignore "concretized symbolic size" error. You will see testNNNNNNN.abort.err file in klee-out-N directory if you find the bug.
    Submit both .abort.err file and .ktest file and content of the symbolic file in the testcase(2pt).
    d. Build the gnupg with gcov

    ```
    $ cd gnupg-1.0.5
    $ cd build_gcov
    $ CFLAGS="-fprofile-arcs -ftest-coverage" ../configure
    $ make
    ```

    e. Initialize coverage result.

    ```
    $ lcov-1.10/bin/lcov -i -z -d .
    ```

    f. Make a program that will run the gnupg with all the testcases generated by Klee in the previous step. Run gnupg with testcases. The executable is build_gcov/g10/gpg.
    g. After execution, make a coverage report using lcov.

    ```
    $ lcov-1.10/bin/lcov -c -d . > coverage.info
    $ lcov-1.10/bin/genhtml -o html coverage.info
    ```

    You will see the overall coverage. Also coverage report of html form will be generated in html directory. Report overall coverage rate.
    h. Submit overall coverage(1pt), your program to run gnupg and klee-out-N directory(2pt).

Part 2 (6pt)

1. Find memory leak with Klee (4pt)
    a. Download and build the program

    ```
    $ curl -O http://www.cs.purdue.edu/homes/kim1051/cs490/proj3/p2.tar.gz
    $ tar xzvf p2.tar.gz
    $ cd p2
    $ sh make.sh
    ```

    b. Add assertions at the end of the program and check if all allocated memories are freed. The program uses my_malloc() and my_free() in misc.c to allocate and release memories. If you need, you can modify those functions also.
    c. Run Klee and find a testcase that cause a memory leak. You will see testNNNNN.assert.err if you use assert() for you assertion and testNNNNN.abort.err if you use abort() in your assertion.

   d. Submit your modified code(2pt) and testcase(2pt) causing a memory leak.

2. Make a small program of which Klee cannot cover all feasible paths. (2pt)
   a. The program should have at least one if statement.
   b. There should be two valid inputs so that one input will cover true branch and the other will cover false branch of the if statement.
   c. When run Klee with the program, Klee should not be able to find one input so that Klee cannot cover either true or false branch of the if statement.
   d. Submit your code and testcases to explore all feasible paths in order to check the path that Klee couldn't cover is a feasible path.

Submission

- By 11/6, 11:59pm, you should finish part 1.
- By 11/11, 11:59pm, you should finish part 2.

Where to get help

Dohyeong Kim(kim1051@purdue.edu) is managing this project. His office hour is Wednesday 10:30am -- 11:30am at LWSN 3151 #4. We recommend you to use the Piazza for sharing your questions and get answers faster. Every day our instructors check the Piazza several times to process students' questions.

Appendix

1. Klee web page http://klee.llvm.org

2. Klee OSDI 2008 paper http://llvm.org/pubs/2008-12-OSDI-KLEE.pdf