CS6888 Program Analysis and its Application
Assignment #1: Due Sept 17th 2PM

Assignment goal: to consolidate understanding and obtain hands-on experience on how fuzzing tools work and how to set them up, where fuzzing excels and where it struggles, how to improve fuzzing performance, and what to do with the information it produces.

This is an **individual** assignment. You are **not to share** your setup, your strategies to assist AFL, your results and analysis, or any other aspect of the assignment with your classmates.

Task 1: Setup

Download, build, install:

1. AFL, a popular fuzzer.
   https://lcamtuf.coredump.cx/afl/releases/afl-latest.tgz
2. afl-cov, a coverage tool for use with AFL.
   https://github.com/mrash/afl-cov
3. The latest version of TCC for your system. This is the system we are going to test.
   http://download.savannah.gnu.org/releases/tinycc/

The tools websites provide plenty of documentation for you to figure out how to get them to run together. At the end of the assignment we provide some tips on how our TA completed this assignment on a Linux box, but keep in mind that these tips do not replace your reading of the tools' documentation. If you do not have a **Linux** box we recommend using *portal.cs.uva.edu,* (note that TCC does not work well on a Mac without significant tweaking and the same may happen under Windows).

Task 2. Run AFL with different strategies

1. Run AFL on TCC for 10 min to get a sense of how it works with the following seed example and the following TCC parameters while collecting coverage information at regular intervals (see report requirements below):

```
#include <tcclib.h>
int main(int argc, char* argv[]) {
  printf("Hello World!\n");
  return 0;
}

/path/to/tcc -I /path/to/tcc/include -o hello -c hello.c
```

2. Implement strategies to get AFL to cover *50% of the lines of code* in TCC. Basic strategies include running AFL for extended periods of time, enriching the seed pool to reach sections that are not covered, and exploring more parameters in TCC being tested besides

program to compile. More and some more advanced strategies are available here: *https://afl-1.readthedocs.io/en/latest/tips.html#performance-tips*

3. Explore the functions not covered in the "tccgen.c", "tccpp.c", and "tccrun.c" files, investigating why they are not covered and what it would take for them to be more extensively covered.

Task 3. Write a report

The report must be **under 3 pages in pdf format**, and include:

1. Specify the machine, OS, and the versions of AFL and TCC used.
2. Plot the line coverage achieved by AFL when running the provided sample program for 1, 5, 10, 30, and 100 minutes (plot x-axis is time, y-axis is coverage). Discuss what you observe.
3. Explain the strategies used to increase the coverage generated by the Fuzzer.
4. Provide the LCOV code coverage report (screenshot of the tool output is sufficient) after implementing those strategies. If you found any failures, please explain how they manifested and what input led to them.
5. Discuss what strategies were the most cost-effective and explain why that was the case.
6. Name the functions not being covered in the "tccgen.c", "tccpp.c", and "tccrun.c" files, and explain why they are not covered and what it would take to cover it.

Extra Credit (1pt): You are given free range to explore any target software you would like to torture with AFL to potentially expose bugs that have not been found by others. The target software, however, must meet the following criteria: 1) be fuzzable by AFL, 2) the software must be open source (you will have to provide a URL) 3) contain more than 10,000 lines of source code, 4) has not been fuzzed before as part of its normal development or to showcase or evaluate a fuzzer (the software cannot be one of the ones in http://lcamtuf.coredump.cx/afl/ or found by doing a google search). This is an opportunity to contribute to the robustness of a piece of software that is of value to you or to a community you care about. I would really like to see unique target programs that are relevant to you. And a warning: do not run the fuzzer on live software or data where it can cause harm! Please include the chosen project URL and a brief description, and a summary of the results obtained in your report following the same format required for TCC.
You are allowed an extra page in the report to include this extra credit.

Submission:

Submit the report through Collab under Assignment #1.
Your grade will be based on the submitted report, the extent to which it covers the required items, the discussion and depth of the strategies explored, the correctness of the results and the insights shown in their explanation. This assignment is worth 10% of your grade.

Additional tips from our TA

On a Linux Box, assuming you follow the directions to get AFL and AFL-cov installed successfully, here are some tips our TA collected when working on this project:

1. Make a copy of the TCC folder and append -cov to the name.
   For example, you should have both ~/Downloads/tcc-0.9.27 and ~/Downloads/tcc-0.9.27-cov
2. Build TCC using AFL. This is the version that AFL will use.
   In the folder containing TCC (for me: ~/Downloads/tcc-0.9.27/), run
   $ ./configure --libdir=$(pwd)/lib --CC={path to}/afl/afl-gcc (or a slight variation: $CC={path to}/afl-gcc ./configure --libdir=$(pwd) /lib)
   $ make clean all
   $ mkdir lib/tcc
   $ cp libtcc1.a lib/tcc/libtcc1.a
3. Build TCC for coverage. This is the version that afl-cov will use.
   In the folder containing the -cov TCC copy from step 1, run:
   $ ./configure --libdir=$(pwd)/lib
   $ make CFLAGS="-fprofile-arcs -ftest-coverage" LDFLAGS="-fprofile-arcs -ftest-coverage"
   $ mkdir lib/tcc
   $ cp libtcc1.a lib/tcc/libtcc1.a

The process our TA followed to run AFL and obtain coverage information:

1. Create a directory for the seed input for AFL, for example: /afl-input.
2. Choose what seed inputs to use. Start with the sample Hello World! C provided. This is a good time to remind yourself what makes AFL tick in the AFL guide: https://afl-1.readthedocs.io/en/latest/quick_start.html
3. Create a directory for the seed output for AFL, for example: /afl-output. This will store the results from AFL.
4. Choose what command line arguments to run with TCC. TCC has many different options. Test the arguments outside of AFL before starting AFL to verify TCC is working as expected.
5. Start afl-cov with the chosen arguments:
   $ {path to}/afl-cov -d{path to}/afl-output --live --coverage-cmd "-I {path to}/tcc/include -o hello AFL_FILE" --code-dir {path to}/tcc-cov (in some machines you may need to install the afl-cov requirements like lcov)
6. In a different terminal, start AFL with the chosen arguments. Replace the input file with @@:
   $ {path to}/afl/afl-fuzz -i {path to}/afl-input -o {path to}/afl-output {path to}/tcc -I {path to}/tcc/include -o hello @@
7. Let AFL run for the desired length of time. When the time has elapsed, press CTRL+C in the terminal running AFL. Once AFL has exited, it may take afl-cov up to another minute to detect this and shut down. Warning: If manually stopped it will NOT write the coverage information to disk!
8. View the coverage reports. Open {path to}/afl-output/cov/web/index.html in a web browser.

To run it under “portal.cs.virginia.edu”

When running on portal.cs.virginia.edu:

1. Use absolute path names. (Don't use ~)
2. Before afl-cov can run, the Perl modules it depends on must be loaded:
   module load perl
3. Before afl-cov can run, you must download and build lcov (https://github.com/linux-test-project/lcov)
   afl-cov must then be run with --lcov-path={path to}/lcov/bin/lcov --genhtml-path {path to}/lcov/bin/genhtml
4. afl-cov will generate an html file - you can either view the raw html source, or you can transfer the file to a local machine. See the UVA CS wiki (https://www.cs.virginia.edu/wiki/doku.php?id=start) for more information.
5. The servers for portal.cs.virginia.edu have a short timeout and will close the connection if there is no activity - this would prevent you from stopping/viewing AFL if you leave it running. This can be fixed by setting up your local SSH client to send a heartbeat (called keepalive) to the server on regular intervals. Using the ssh client in your terminal, create ~/.ssh/config with the following

   ```
   Host *
     ServerAliveInterval 120
     ServerAliveCountMax 5
   ```

   For more information check: https://www.a2hosting.com/kb/getting-started-guide/accessing-your-account/keeping-ssh-connections-alive for more info.
6. When compiling TCC on portal.cs.virginia.edu - you may see an error
   /bin/sh: makeinfo: command not found.
   That is okay - makeinfo is used to generate documentation during the compiler process. portal does not have this software, but if you made it to that step then the tcc binaries have been generated.