System Configuration:

1. Login to change some system settings
2. Then on the terminal run cat sys_tuning.sh
3. ./sys_tuning.sh on the terminal (this step is deal with system frequency and deal with process time outs)
4. Exit from the root

Stage 1: Instrumentation and build

Start the execution of the tool , path is cd workspace/yfuzz/testcases/openssl_solution

1. Go to the openssl folder:
   cd workspace/yfuzz/testcases/openssl_solution
2. Uncomment injected forking points in openssl.c source file
3. Use ./build to compile the protocol

Stage 2: Configuration and Fuzzing

To configure, we have to use cd ../.. to go back to yfuzz folder.

1. Go to yFuzz folder: cd workspace/yfuzz
2. Modify the config.h file (vim config.h)
3. Compile using make (directory: workspace/yfuzz$ make)
4. Do ./runssl.bash _solution to execute the tool

Stage 3: Exercise

Goal: Add a second forking point in the yFuzz tool and repeat stage 1 and stage 2

cd workspace/yfuzz/testcases/openssl_solution

In this stage,

- The audience will be given a hint – second forking point should be added after server writes packet 2 and right after the consistency check
  cd workspace/yfuzz/testcases/openssl_solution
  1. In openssl.c source file, add the forking point right after server sends packet 2, and after the consistency check i.e, if(TPstate==pstate)
  2. Use ./build to compile the protocol
- After adding the second forking point, the next step is to configure the parameters in the config.h header file and run the yFuzz.
- Go to yFuzz folder: cd ../..
- Modify the config.h file (vim config.h)
- Compile using make (directory: workspace/yfuzz$ make)
- Do ./runssl.bash _solution to execute the tool

Forking point format:

```
#ifdef __AFL_HAVE_MANUAL_CONTROL
                    __AFL_INIT();
#endif
```

Parameters in config.h file:

- #define PROCEED_COE 2 //set the probability of moving forward the fuzzing state (1-10)
- #define MIN_CYCLE_TO_PROCEED 0 // will finish one cycle before proceed; set minium cycles to complete before changing states (>=0)
- #define MAX_CYCLE_TO_REGRESS 1 // will finish one cycle then regress; sets the maximum cycles a fuzzing state can have (>=0)
- #define Q_MAX_PATHS 100 // sets the max number of test cases a fuzzing state can have (>=0, e.g. 200)

Commands we used:

1. ls – to list the files and folders in the directory
2. cd – to change the directory
3. vim – to edit the file
4. esc and :wq – exit and save the file
5. make – compile the program

Note:

- Build compiles all source files in the project.
- Make compiles those that have changed since the last make or build.

6. ./<file_name> - to execute the program
7. cd ../ - goes the previous directory