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\_stage1

1. Go to the openssl folder:

cd workspace/yfuzz/testcases/openssl\_stage1

2. Inject the forking points in openssl.c source file vim openssl.c

3. Save the source file and quit :wq

4. Make sure to set up the input and output folders

5. cat build

6. Use ./build to compile the protocol

Stage 2: Configuration and Fuzzing

To configure, we have to use cd ../.. to go back to yfuzz. If you miss adding the forking points in the stage 1, you can directly go to the stage 2 folder

cd workspace/yfuzz/testcases/openssl\_stage2

1. Go to yFuzz folder: cd workspace/yfuzz
2. Modify the config.h file (vim config.h)
3. Save the config.h file and quit (:wq)
4. Compile using make (directory: workspace/yfuzz$ make)
5. Type cat runssl.bash and
6. Do ./runssl.bash $version to execute the tool

\* version is defined as “\_stage1 or \_stage2 or \_stage3” depending ont the stage

Stage 3: Exercise

Goal: Start with clean slate openssl.c and config.h to run yFuzz tool

cd workspace/yfuzz/testcases/openssl\_stage3

In this stage,

* The audience will be given a clean slate openssl.c source file with 1 forking point inserted and the task is to figure out the second forking point with the instructions we mention and follow the same procedure from stage 1

1. Go to the openssl folder:
2. cd workspace/yfuzz/testcases/openssl\_stage3
3. Inject the forking points in openssl.c source file, add second forking point.

vim openssl.c

1. Save the source file and quit :wq
2. Make sure to set up the input and output folders
3. cat build
4. Use ./build to compile the protocol

* After adding the second forking point, the next step is to configure the parameters in the config.h source file. The audience can change the values within some range defined in the source file and visualize the output layout.

Stage 2: cd workspace/yfuzz/testcases/openssl\_stage2

* Go to yFuzz folder: cd workspace/yfuzz
* Modify the config.h file (vim config.h)
* Save the config.h file and quit (:wq)
* Compile using make (directory: workspace/yfuzz$ make)
* Type cat runssl.bash and
* Do ./runssl.bash $version to execute the tool

\* version is defined as “\_stage1 or \_stage2 or \_stage3” depending ont the stage

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)