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. If you miss adding the forking points in the stage1, you can directly go to the openssl\_stage2 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. Compile using make (directory: workspace/yfuzz$ make)
4. Do ./runssl.bash $version to execute the tool

\* version is defined as “\_solution or \_stage2 or \_stage3” depending on the stage

Stage 3: Exercise

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

cd workspace/yfuzz/testcases/openssl\_solution

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: cd workspace/yfuzz/testcases/openssl\_solution
2. In openssl.c source file, add the forking point right after server sending packet 2, and state consistency check. Add it in the if statement: if(TPstate==pstate).
3. 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. The audience can change the values within some range defined in the header file, and then visualize the output layout.
* Go to yFuzz folder: cd ../..
* Modify the config.h file (vim config.h)
* Compile using make (directory: workspace/yfuzz$ make)
* Do ./runssl.bash $version to execute the tool

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

If you have trouble in adding new forking point in this stage, please go to the openssl\_stage3 folder:

cd workspace/yfuzz/testcases/openssl\_stage3,

the second forking point has already been added into openssl.c source file in this folder.

Fokring 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.

1. ./<file\_name> - to execute the program
2. cd ../ - goes the previous directory