CS 5493 Lab 1 - Beginner Fuzzing

Sam Dutson - u1406827

Overview(What I'm Fuzzing):

Since this is my first time using AFLplusplus, I decided to fuzz the 'vulnerable.c' program that's provided in the afl-training repo.

You can find the repo here: https://github.com/mykter/afl-training/tree/main/quickstart

I made heavy use of the README file in the repo. It walked me through the steps of fuzzing the program.

User-Configurable Features:

1. **AFL_USE_ASAN**: This feature enables address sanitization when compiling the target program. This feature makes bugs reliable and reproducible. It also makes it easier to triage the bugs. **However,** this feature does decrease speed.

   Use:
   - CC=afl-clang-fast AFL_USE_ASAN=1 make clean all
   - afl-fuzz -i inputs -o out/out_use_asan ./vulnerable

2. **AFL_NO_FORKSRV**: This feature forces AFL++ to fully restart the program for every test case(Rather than just forking the process). Enabling this feature should greatly slow down the program. This will show how important the forkserver is for AFL++.

   Use:
   - CC=afl-clang-fast make clean all
   - AFL_NO_FORKSRV=1 afl-fuzz -i inputs -o out/out_no_fork ./vulnerable

3. **-p(Power Schedules)** ' **-p fast, -p explore':** The power schedule determines which inputs get fuzzed more aggressively. The two options that I'll be using will be '-p fast'(focuses on speed) and '-p explore'(focuses on coverage).

   Use:
   - CC=afl-clang-fast make clean all
   - afl-fuzz -p fast -i inputs -o out/out_fast ./vulnerable
   - CC=afl-clang-fast make clean all
   - afl-fuzz -p explore -i inputs -o out/out_explore ./vulnerable

Experiment/Results:

For each experiment I ran the fuzzer on the './vulnerable' executable for 3.5 minutes. I extracted info from the 'fuzzer_stats' file for each execution.

Note: I've added all fuzzer output files to the following repo if you'd like to take a closer look: https://github.com/sdutson/CS5493_Lab1

Fuzzer Mode	Total Executions	Executions Speed(exec/sec)	Saved Crashes	Edges Found	Bigmap coverage
Default AFL++:	1730456	8240.04	11	110	92.44
AFL++ with AFL_USE_ASAN:	1584529	7514.82	15	110	92.44
AFL++ with AFL_NO_FORKSRV:	91696	435.79	7	110	92.44
AFL++ with -p fast:	1565882	7423.32	19	110	92.44
AFL++ with -p explore:	1562138	7416.57	12	110	92.44
ALF++ with AFL_USE_ASAN AND -p fast:	388390	1843.37	21	110	92.44
AFL++ with AFL_USE_ASAN AND -p explore:	388696	1847.05	22	110	92.44

Analysis:

Default AFL++:

- Don't have too much to note for this option. I mostly just added it as a good baseline for comparison.

AFL++ with **AFL_USE_ASAN**:

- This setup doesn't differ too much from the default AFL++ run. The two differences that do exist are the Total executions and Saved Crashes. My hypothesis is that both of these are tied to the fact that AFL_USE_ASAN enables address sanitization(This adds overhead but also exposes memory bugs).

AFL++ with **AFL_NO_FORKSRV**:

- The biggest thing that I noticed with this setup was just how few executions there were. The average executions per second was only ~6% the speed of the Default AFL++ option. The root cause of this is almost definitely the time it takes to restart the process each time(instead of just calling 'fork').
- This option also found fewer bugs than the other executions. This is likely a direct result of the smaller number of total executions.

AFL++ with **-p fast**:

- I'm very surprised with the output from this setup. While it did find more bugs than the default option, the number of executions was smaller than for default AFL++. My hypothesis is that this has something to do with the fact that the 'vulnerable.c' program that I am fuzzing is quite small(There are fewer options for possible things to fuzz). This makes the benefits from prioritizing speed less noticeable.

AFL++ with **-p explore**:

- This setup had very similar output to the default AFL++ run. Again, my hypothesis is that this is due to how small 'vulnerable.c' is. For a much larger program, I would expect this option to have a slower execution time but better overall code coverage.

ALF++ with **AFL_USE_ASAN** AND **-p fast**:

- This option ran much slower than the default AFL++ run, the AFL++ with AFL_USE_ASAN run, and the AFL++ with -p fast run. However, this option did find significantly more crashes.

AFL++ with **AFL_USE_ASAN** AND **-p explore**:

- This option had a very similar output to the 'ALF++ with **AFL_USE_ASAN** AND **-p fast'** run option.

General Notes:

- ALL run options had the same values for 'Edges Found' and 'Bitmap Coverage'. I expected some variation between the different options. My hypothesis for this is that the 'vulnerable.c' file that I fuzzed had a few branches that are EXTREMELY hard to hit.
- To test this I ran the default AFL++ option on the 'vulnerable.c' file for 26 min to see if I could get higher Bitmap Coverage. I got the following values:

Fuzzer Mode	Total Executions	Executions Speed(exec/sec)	Saved Crashes	Edges Found	Bigmap coverage
Default AFL++ with 30 min runtime	11601402	7434.44	16	110	92.44

- These results were also very surprising. Even with the longer run time, almost all the stats for the experiment stayed the same(crashes, bitmap coverage, ect). This leads me to believe that there are some branches of the code that are only accessible for VERY long runs of fuzzing systems.

Sources:

- I made heavy use of the documentation in the following repo: https://github.com/mykter/afl-training/tree/main/quickstart