Assignment 4 - CMPT 473

Projects Analyzed

Used with LibFuzzer

- ❏ Project name - Newsbeuter
- ❏ Github link - https://github.com/akrennmair/newsbeuter
- ❏ Openhub link - https://www.openhub.net/p/newsbeuter
- ❏ Size of the code base - 33,120 (According to Openhub)
- ❏ Build time to compile and link an executable from source code - ~70 seconds
- ❏ Execution time for the test suite - ~0.66 seconds

Used with FuzzFactory

- ❏ Project name - TCPdump
- ❏ Github link - https://github.com/the-tcpdump-group/tcpdump
- ❏ Openhub link - https://www.openhub.net/p/tcpdump
- ❏ Size of the code base - 271,088 (According to Openhub)
- ❏ Build time to compile and link an executable from source code - ~36 seconds
- ❏ Execution time for the test suite - ~4.8 seconds

Reflection

LibFuzzer: Process and Challenges

Starting with LibFuzzer. For our first step, we had to choose a project to be tested. We ended up choosing Newsbeuter, an RSS feed reader for the terminal. Our next step, which required some effort, was to install all of the dependencies used by the tool, of which there were many of; nonetheless, the readme file provided a clear list of all the dependencies required, making the downloading and installation process less troublesome. Once we had all that taken care of, then came the building of the project and running of the test suite. When we confirmed the project compiled properly, then came the first real challenge, finding a perfect function to fuzz.

Due to how fuzzing with LibFuzzer works, we had to look for a function to be used with the test driver that we were going to write. Although this sounds simple, we had to look through many functions before we eventually chose a string formatter function named do_format inside formatstring.h. We chose this because the implementation involved a large amount of string manipulation and the fact that it took in only a string made it simple to develop our test driver as it only needs to turn the array of bytes into a string.

The final step, which was also the most time consuming and challenging step, was the compilation step. As we chose to compile the test driver directly through the terminal, we had to link the fuzzer to the implementation file of the function along with every other file that file depends on including external dependencies. A lack of experience with Clang and misunderstanding of how some of the flags worked made trying to compile the test driver one of the more time-consuming processes for Libfuzzer. After a while of searching on the web and learning how Clang and file linking worked, we eventually got the test driver to compile.

FuzzFactory: Comparing the Two Techniques

Since no crashes or hangs were observed when we ran the FuzzFactory on tcpdump, we did not notice any difference in that aspect; however, looking at the executions done, executions per second, total paths taken, total paths favored, and total paths found, we did notice a difference between the two techniques.

In our results, we found that when only the cmp domain was used, only 5707654 executions were completed, 183.88 executions per second, 1015 paths taken, 223 paths favored, and 1001 paths found; this was compared to the 6848277 executions, 221.46 executions per second, 1033 paths taken, 216 paths favored, and 1019 paths found when both the cmp and mem domains were used. Furthermore, before we ended the fuzzing process, we noticed that when only the cmp domain was used, there were 95 timeouts, 15 of which were unique, compared to only 60 timeouts and 11 of which were unique when both the cmp and mem domains were used.

More Challenges and Strengths and Weaknesses of FuzzFactory and libFuzzer

Here are some more challenges we faced along with some strengths and weaknesses of the fuzzing tools we used.

First of all, getting the fuzzer to start working. Compared to LibFuzzer, FuzzFactory was more work. As mentioned in the LibFuzzer section, LibFuzzer requires a test driver to be written for every single function that the user wants to be fuzzed; this can be a strength, but also a weakness of the fuzzer. On one hand, having the fuzzer pass in many different inputs into a single function can and did help us find a bug that may have otherwise been missed if only a single input was provided to that function; on the other hand, writing a test driver for every function can be time-consuming and burdensome. Furthermore, writing this test driver can be difficult as we found in our experience. In some cases, it can be unclear how to translate the given data into something representative for the function under test especially if the argument being passed in is some kind of complex object.

But even before we could start writing the test driver, we needed to prepare and find a proper project and a proper function within that project. In our experience with Libfuzzer, this was one of the most challenging tasks. A lot of time was spent examining many repos only to discover some kind of issue with them that made them unfit for fuzzing or the assignment. These issues include the repo lacking a test suite of any kind, use of complex objects that made translation from a set of bytes difficult, many complex functions having preconditions for the data passed in which will throw an exception if not met, or the project would not compile properly. It was only until hours passed before we found a perfect project, the project that our results came from. From this, we can say that the need to write a test driver is a great struggle and a great weakness of the tool.

With regards to finding a perfect project for fuzzing, the same can be said when seeking for a project to be used with FuzzFactory. When we tried using Newsbeuter, the project that we used for LibFuzzer, with FuzzFactory, we simply could not get the fuzzer to explore more than one path. After many tries with modifying files and creating different test drivers, we decided to try fuzzing with a different project. Even then, it was not until after many trials with many different projects, did we find a project that could be used with FuzzFactory. This leads us to a weakness of FuzzFactory. FuzzFactory only works with programs that take in an input either from a file or stdin; and sadly, it turns out that the main program of Newsbeuter does not take inputs from files or stdin, as a result, we were not able to use the same project for both fuzzing tools and therefore, was not able to do result comparisons.

However, once we did find a project that could be used with FuzzFactory, a provided test driver meant that we did not have to write one ourselves, therefore relieving the burden. But unlike LibFuzzer, FuzzFactory requires an initial set of input files; this we think is a strength, but also a weakness of the fuzzer. By providing an initial set of test inputs, we are giving the fuzzer an example of what type of inputs the program expects, therefore the fuzzer can based on the input files, generate only useful, mutated inputs, leaving out the non-relevant, useless inputs, allowing for a more effective fuzzing process. However, because we are giving the fuzzer initial inputs, we basically tell the fuzzer to generate inputs related to the input files given, thereby restricting the range of inputs tested. As for whether the generation of inputs based on given input files was reflected in our results, yes, the chosen input files did affect our results. As mentioned, FuzzFactory creates inputs using the files provided; therefore, what type of files the user gives, will cause the fuzzer to generate different inputs, therefore affecting the results. In our case, we simply chose a subset of the files in the tests folder of the original program to be used by the fuzzer.

Finally came the issue of running the fuzzing tools. Starting with LibFuzzer. As mentioned in the LibFuzzer section, due to our lack of knowledge with regards to file linkage, Clang and misunderstanding of how flags work, compiling the test driver was a struggle. While this is not a direct issue of the fuzzing tool itself, requiring the user to figure out what files to link and how to write the linkage and flags in order to compile the test driver is a weakness of the tool. This is reflected in the invocation file, but not in the results because the purpose of the invocation is to tell the compiler what files to link. As for FuzzFactory, once we found the perfect project that worked perfectly with FuzzFactory, running the fuzzer was for the most part, easy. Figuring out how to compile the project with FuzzFactory, how to write the command and just fixing the issues the fuzzer complained about took a bit of work and required a bit of searching around on the web, but that was relatively quick. Again, what was done with FuzzFactory here is not reflected in our results because all that was done here is to get the fuzzer running correctly.

Assignment 4 - CMPT 473

Bugs Reported by Fuzzing Tools

Of the two projects analyzed, we only encountered one bug. This bug occurred when LibFuzzer was fuzzing the newsbeuter project. Fuzzing with that tool will always eventually lead to a stack-buffer-overflow detected by the address sanitizer. We are not experts in address sanitizer so we cannot fully decipher the error output but the fact that all the driver does is pass a string into the function and the function accesses the invalid address, this leads us to believe this is a legitimate bug with the program.

As for why there were fewer than 2 bugs found, in the first program, we tested the Newsbeuter project with LibFuzzer. By default, LibFuzzer stops after the first crash, therefore, once we hit the stack-buffer-overflow issue, the fuzzing stopped, so we only found one bug there. In our second project used for testing, tcpdump turns out to be a widely fuzz tested project, therefore within our two four hour fuzz tests, it is understandable if there were no bugs or crashes discovered.

We do want to note however, that when we ran FuzzFactory with the cmp domain, 95 timeouts were observed, 15 of which were unique; and when we ran the fuzzer with both cmp and mem domains, 60 timeouts were observed, 11 of which were unique. Of the 95 and 60 timeouts observed, none exceeded the timeout threshold by a great enough amount to be considered to be a hang by the fuzzer, therefore, none of the timeouts observed were real bugs.