Assignment 4

Using Program Analysis Tools

For this project, you will gain experience using dynamic analysis tools on real world software.

You will work in groups of two or three students, and the requirements will scale with the group size.

Overview

As a group, you will apply two types dynamic analysis tools we examined in class to open source projects and report on the results. One set of tools will look for bugs in a preexisting set of executions, while the other will create new executions/tests in an effort to find bugs. Note that not all tools will work with all programs, so the type of program that you analyze must be compatible with the tools that you use.

Detecting bugs in an existing test suite

Possible dynamic analysis tools for this assignment include:

- Valgrind
- Clang/GCC Sanitizers

Recall that dynamic analysis tools analyze a single execution at a time. Thus, it is common practice to integrate the dynamic analysis tools with the test process for a project in order to detect potential bugs over a set of predetermined executions. In order to apply your selected tool to a given open source project, you shall integrate the dynamic analysis tool into the existing test suite and automated testing infrastructure for the project and run the analysis over every execution in the test suite. If you are interested in using an unlisted dynamic analysis tool, *ask to make sure that it is appropriate*. Your write-up for the assignment should include the challenges you faced during this process, as well as your approaches for overcoming them. You should also report any errors indicated by the analysis. For groups of k members, the groups should also explain whether k of the errors found were real bugs or not. If fewer than k bugs were found, then all errors should be explained. *Include the full instructions for reproducing your results based on files that you had to modify and how.*

Be careful. A very common mistake is to run `valgrind` on `make` instead of running it on the project you wish to analyze. We discussed issues like this in class, and I expect you to not make this mistake.

Fuzzing

Possible fuzz testing tools for this assignment include:

- american fuzzy lop
- libFuzzer

If you wish, you may alternatively try out one of these tools, but I won't provide support for them:

- AFLFast
- VUzzer

In contrast to the previous set of tools, fuzz testing tools attempt to generate new inputs (and thus executions) of a program and look for general correctness issues as those new tests run. They do not need to be integrated into a test suite, but you sometimes need to construct a test harness for particular functions of interest, as we saw in class. You will use one of these fuzz testing tools to test the behavior of a program or library when reading in some sort of input from the user. You should run the fuzzer for *at least 8 hours*. Ideally, and if you want more interesting results, you would let it run for longer. You can use the `screen` command to log out of a machine while your tools continues to run and then log back in later to see the results. If you are interested in using an unlisted fuzz testing tool, *ask the instructor to ensure that it is appropriate*. Your write-up for the assignment should again include the challenges you faced during this process, as well as your approaches for overcoming them. You should also report any errors indicated by the analysis. For groups of k members, the groups should also explain whether k of the crashes/hangs found were real bugs or not. If fewer than k bugs were found, then all discovered errors should be explained. You should also explain *why* fewer than k bugs were found if possible. This may depend on both the tool and the observed results.

Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder

In addition to this write-up, you should also submit any test harnesses, invocations, and the full set of constructed outputs that cause problems as well as the overall statistics. For AFL, this means that you should submit the `crashes/` and `hangs/` subdirectories of the output directory as well as the `fuzzer_stats` file. For libFuzzer, you should include any test files corresponding to failures along with the output of the overall testing process.

Be careful. `afl` may warn you that the program you are analyzing is not instrumented or that you are only identifying one path after many executions. These are signs that you are not correctly running `afl` to fuzz the program of interest. Again, I expect you to not make these mistakes.

Selecting Projects to Analyze

For software that you analyze should be an open source project of some sort. Any analyzed project should contain at least 4000 lines of code and must include an established test suite and automated test process. You are also free to analyze two different projects, one for each type of tool. Once again, you are free to consider different projects listed on www.github.com, www.sourceforge.net, www.openhub.net, www.gnu.org, or other major collections of open source software. If you have questions about the suitability of a particular project, please *ask*. Finally, do not select one of the programs that I already showed you in class.

Once again, you should identify and consider:

1. Identification of the open-source project.
2. Identification of the supporting organization.
3. Size of the code base.
4. Build time to compile and link an executable from source code.
5. Execution time for the test suite.

Again, include this information in your report.

Submission

As a group, you should reflect on the challenges faced, effort required, and either potential or recieved benefits of the tools you used for the projects you examined. What are the strengths and weaknesses of the different types of dynamic analysis tools that you used? Are these reflected in your results? Why or why not? How? How might these compare to the strengths and weaknesses of static analysis tools? You should form and *justify* an opinion as to which was more useful for the project(s) you examined.

The assignment is due 11:59 pm on Wednessday, March 21.

2018 Nick Sumner
HTML - CSS - Jekyll