Exploratory Assignment

17-712: Fantastic Bugs and How to Find Them (Spring 2025)

Before embarking on projects involving domain-specific program analysis tools (as you will be doing for your final course project), you should gain some experience working with general analysis tools designed to help find bugs in programs in a domain-agnostic way. This assignment is designed to provide you with such hands-on experience, while providing you with ample freedom in choosing your own analysis technique, tool, and target programs.

This assignment can be performed individually or in teams of two.

Selection Due Date: February 04, 2025 by 11:59pm
Report Due Date: February 20, 2025 by 11:59pm

Task

The overall goal of this assignment is to run an automated bug-finding tool on at least two programs and report your experience. It is important to document your process and present insights about the strengths and weaknesses of the chosen tool, as well as the manual effort required in using it to find bugs.

Specifically, you need to do the following:

1. Pick a domain-agnostic bug-finding tool of your choice. It can use static or dynamic analysis, and it may or may not be any of the tools that we have explored in class. Examples include (but are not limited to):
    - Cppcheck
    - Spotbugs
    - Infer
    - ErrorProne
    - Checker Framework
    - Clang Static Analyzer
    - AFL++
    - LibFuzzer
    - JQF
    - Java PathFinder
    - KLEE
    - S2E
    - CBMC
    - Other static analysis, dynamic analysis, fuzzing, or symbolic execution tools

2. Pick at least two target programs to analyze. These should be open-source programs hosted on GitHub or similar, and in regular use. A good rule of thumb is that the project should have at least 100 stars (1000+ stars is even better) and have a code commit within the last 6 months (last 1 month is even better).

   You can pick these programs using some judgment about where your chosen tool will be most effective. Think about requirements for your tool, such as whether you need to be able to build the code, run tests, etc. and whether the chosen tool supports all the languages and frameworks used in the target program.
   Hint: Try to find examples of new bugs discovered by the tool you have chosen, and use that as a guide to pick target programs from that list or similar programs.

3. Run your chosen analysis tool on the sample program, while observing the output in detail. Try to understand what parts of the target program your analysis tool is analyzing, what parts it is ignoring/missing, whether you are getting warnings that seem like false positives, etc. Be ready to dive into the relevant source code files to make sense of analysis results as required. If your analysis tool was not configured properly (e.g., perhaps you need to provide a specific entry point or include certain library/header files), use this interaction to reconfigure it and try again.

4. Demonstrate how the tool can find bugs in each of the chosen target programs. You can do this in any of the following three ways:
   - If the tool finds new bugs in the latest version of the software, then congratulations! You should file a new bug report in the project's bug tracker and follow up with the developers to help them fix it. If no new bugs are found in the latest version, then either:
   - Try to re-discover a previously fixed bug by checking out a buggy release version or a specific commit where an old bug was discovered. If you chose the target program because your tool has already been successful at finding bugs in that project in the past, then you can use this information to select an old buggy version. Otherwise, you can search through the project's bug database / pull requests to find an interesting and relevant bug that you think your chosen tool could have found. If you identify such a historical bug but your tool cannot find it when run on the old version, document why you think the tool fails to do so.
   - If the above does not work for you, then manually inject a bug in your target programs by editing the source code and simulating a programmer error (e.g., remove a bounds check or edit loop boundaries). Then re-run the analysis tool to "discover" the injected bug. If this doesn't work the first time, document why you think your injected bug was not discovered by your tool. Modify the injected bug until your tool is successful.

Deliverables and Milestones

1. **Declare your selection of tool and target programs (due in 1 week: Feb 4th, 2025).** We will simply ask you to post the name and URLs of the chosen tool and target programs of your choice, along with a 1-sentence summary of why you chose these. The reasons could be objective (e.g., because you wanted to reproduce the discovery of a specific bug) or personal (e.g., because you are doing research in the area related to the tool or target program).

   There are no points for this milestone, but its purpose is to allow the teaching staff to audit your selection and guide you in case we feel there may be risks that need to be mitigated (e.g., if we think your selection will lead to many configuration challenges that would take a lot of time and not provide much learning value).

2. **A final report (due in 23 days: Feb 20th, 2025)**. In this 2-4 page report, you will provide:
   a. [5 pts] A basic description of your chosen bug-finding tool and how it works (overview only, not technical details).
   b. [5 pts] Links to your chosen target programs with summary stats about their size and popularity.
   c. [10 pts] A description of your experience running the analysis on your target programs. Of particular interest are the manual steps you had to perform in setting up the analysis such as identifying source files and libraries, configuring build systems, choosing entry points, running tests, selecting sample inputs, etc. How long did these steps take? Were any of these steps unexpected or particularly non-obvious? In hindsight, would you do anything differently if you had to run the analysis on another project?
   d. [10 pts] A description of your experience interpreting the results of the analysis tool. Of particular interest are the interactions you had with the tool such as understanding what parts of the code were analyzed and what were not, figuring out when to stop or reconfigure the analysis, connecting analysis results to source files, understanding reports of potential bugs and debugging them, etc.
   e. [10 pts] A description of your experience discovering bugs in the target program. Did you find new bugs, old bugs, or artificially injected bugs? If the latter two, how did you choose the bug to identify/inject? Did you come across any bugs that the analysis tool could not identify? If so, why do you think that is? When a bug is reported, how easy is it to identify the root cause of the problem?
   f. [10 pts] A comparison between your experiences in running the tool on two different target programs. Was the process very similar or different? Did you have to do any program-specific configuration? Did you have to understand the target program or its domain in depth? How easy is it to run the analysis tool on a new target?