EECE 513 Assignment 2 - Fault Injection

1. Outline

In this assignment, we will be learning how to perform a fault injection experiment for real world applications, and protect them from faults. The goal is to understand the error resilience characteristics of applications from hardware faults, and in particular, to be able to understand the propagation of the fault in the program, and mitigate its effects.

2. Basic knowledge of fault injection

The process of a injecting a fault is as follows: during each execution run of a program, inject a single fault in the program, and wait until the program finishes to see if the fault affects the result of the program. In our context, a fault is a bit-flip on the value of a program register or instruction operand. The outcome of a fault injection can be categorized as benign (nothing happens), crash (system crash or program exception), silent data corruption or SDC (incorrect output) and hang (the application times out)

3. Tools

We will be using LLFI [1], a fault-injection tool developed by the dependable systems lab at UBC. LLFI is a compiler based fault injection tool, that injects faults into the LLVM bitcode of the application source code. The faults can be injected into specific program points, and the fault propagation can be easily tracked back to the source code. For more information on the fundamental concepts and basic usage of LLFI, please refer to the online wiki [1]. Since LLFI is built on top of LLVM, you may also want to read reference [2] on LLVM bitcode on which LLFI operates.

4. Experiment

The first step after installing LLFI is to set up the LLFI environment, pick some simple programs (e.g. programs under LLFI/test_programs) and do some trial runs of fault injection. Refer to the wiki [1] for this.

Then, you need to choose one benchmark (e.g. from Rodinia benchmark suite [2]) and conduct a real fault injection campaign over it. The number of runs should be set to 3000.

You should should so the following:

1. Configure your fault injection, i.e. to specify different types of instructions to inject, or to specify different types of registers to inject. For example, you can choose to only inject into ADD instructions by setting up the instruction type to add in your input.yaml. Another example is to choose only operand registers to inject. There is no requirement for you on choosing any

particular type of instructions, but you need to configure for at least one combination of instruction selector and register selector, and conduct the fault injections. (e.g. ADD instructions and destination registers, or SUB instructions and source register)

2. Fix your registers and configure “all” for you instruction type in input.yaml and conduct the fault injection experiments again.

3. Compare the results of your fault injection experiments and provide your insights.

Next step is to pick one case where the outcome is silent data corruption, and try to trace it to see how it ends up as an SDC from injecting the fault. Once we understand this, we need to propose a little instrumentation in the source code of the program, to detect this kind of faults. Verify by injecting a targeted fault and seeing if your instrumentation detects it.

Tips: the case you pick should be easy to reproduce via the fault injection, and the detection mechanism that you propose should be noticeable in your verification.

5 Deliverables

1. The error resilience characteristics of the application you pick. (i.e. the percentage of each type of outcome: Crash, hang, SDCs and benign)
2. The report of your trace for one fault leading to an silent data corruption (SDC).
3. Your proposal to detect this fault, the corresponding rationale and, the verification results.

References

[1]: https://github.com/DependableSystemsLab/LLFI
[2]: http://lava.cs.virginia.edu/Rodinia/