Finding the Right Consistency Model

A comparison of the different consistency models in S2E for testing

Mariana D'Angelo
University of Toronto
Toronto, Canada
mariana.dangelo@utoronto.ca

Dhaval Miyani
University of Toronto
Toronto, Canada
dhaval.miyani@utoronto.ca

I. Introduction

S2E is a symbolic execution platform that operates directly on application binaries for analyzing the properties and behavior of software systems. It analyzes programs in-vivo (the whole environment) within a real software stack (user program, libraries, kernel, drivers, etc.) rather than using abstract models of these layers. It is commonly used for performance profiling, reverse engineering of proprietary software, and finding bugs in user-mode and kernel-mode binaries [1].

There are six different consistency models in S2E, described below. A model is consistent if there exist a globally feasible path through the system for every path explored in the unit. A unit is the block one wishes to analyze and the environment is the rest of the system.

- *Strictly Consistent Concrete Execution (SC-CE):* No symbolic execution in unit or environment.
- *Strictly Consistent Unit-level Execution (SC-UE):* Unit is symbolically executed while the environment is executed concretely.
- *Strictly Consistent System-level Execution (SC-SE):* Unit and environment are executed symbolically – this is the only model that executes the environment symbolically.
- *Local Consistency (LC):* Similar to SC-UE, but it adheres to constraints that the environment/unit API contracts impose on return values.
- *Relaxed Consistency Overapproximate Consistency (RC-OC)*: Similar to LC, but it ignores the constraints from the environment/unit API contracts.
- *Relaxed Consistency CFG Consistency (RC-CC):* Similar to SC-UE, but like static analysis it can explore any path in the unit's inter-procedural control flow graph (even infeasible ones).

The goal of this experiment is to evaluate the performance of the consistency models in the S2E system when finding different types of bugs. We are interested in investigating whether we can identify program "features" (e.g., program length, bug type, etc.) that pose challenges for certain consistency models or present benefits. Some software systems are only exposed to certain kinds of bugs (e.g., a batch processing script will not be exposed to concurrency bugs). This observation leads us to believe that a given consistency model may fit specific a application domain better than others.

As different consistency models treat the environment differently we expect those that symbolically execute the environment will perform slower and have higher memory consumption [2]. As stated previously certain consistency models permit inconsistency in the environment and we expect that bugs which manifest on changes in the environment will not be detected by these models. The SC-SE model suffers from path explosion [1] and as such we expect it to have the worst performance overall, though it should catch bugs given unlimited time and resources.

Although [1] recommends the use of the RC-CC model we expect that there will be certain cases in which other models outperform it. If this holds true, we will perform an analysis to determine what the causes are and if certain program features trigger the performance differences.

II. Key Steps

1) Select three programs that exhibit different types of bugs (e.g., Arithmetic bugs like divide by zero, resource bugs like null pointers or buffer overflows, etc.). These programs will be open-source with either well-known bugs or bugs we have injected. Programs will be carefully selected because of S2E's performance limitations related to program size. Metrics for each sample program (e.g., LOC, number of conditional statements, number of system calls, etc.) will be gathered at this point.
2) Determine the measurement methodology for the metrics used to evaluate different consistency models. Tentatively these metrics will include time to find the bug and the memory consumption.
3) Run each of the consistency models on each of the applications with some cut-off time (exact value to be determined) and gather measurements for the aforementioned metrics.

4) Gather and analyze the performance results for each consistency model along with the sample program metrics to identify any trends.

REFERENCES

[1] V. Chipounov, V. Georgescu, C. Zamfir, and G. Candea. Se- lective symbolic execution. In Workshop on Hot Topics in Dependable Systems, 2009.

[2] C. Cadar, D. Dunbar, and D. R. Engler. KLEE: Unassisted and automatic generation of high-coverage tests for complex systems programs. In Symp. on Operating Systems Design and Implementation, 2008.