Concolic execution supporting IPC

Joonwon Choi, Kyel Ok, Chanwoo Chung

November 1, 2014

1 Problem:

Concolic execution is a powerful tool to explore bugs in a system that are difficult to encounter in a normal workflow. In complex systems that modify their behavior based on the input value, some special inputs or conditions may execute a part of the system that's rarely used. Thus, an adversary can exploit bugs in less explored parts of a system by carefully formulating the input to the system. While it is both difficult and tedious to test a system manually, concolic execution provides an automated way of preventing such attacks by testing many branches of the system workflow and catching bugs that are rarely encountered otherwise [1].

While the benefits of using concolic execution are clear, there are challenges to using it on processes that interact with resources outside of their scope. Resources such as file systems, external databases, and network packets provide inputs that a concolic execution framework has no control over. These inputs may prevent the concolic execution framework from entering some branches of the workflow, weakening the ability to achieve a high code coverage.

One way to deal with external resources is capturing all legal values of the external resources by modeling the environment with a good understanding of the semantics. However, such approach takes human intervention even for the simplest resources, and for complex resources, i.e., packets from another system, it may be impossible to model the input without carefully analyzing the source system that generated the packets.

For this project, we are particularly interested in using concolic execution in the extreme case of complex external resources, in which there are several independent processes that are communicating with each other and serving as an external resource to one another. Such case of inter-process communication (IPC) is commonly found in many systems; one example would be the full zoobar application (without the RPC removed) and a real world example would be a complex robotic system that has many processes, e.g., planner, controller, perception, user-interface, etc, interacting with each other to achieve autonomy.

2 Proposed Solution:

For the scope of the project, we do not attempt to design a concolic execution framework that can deal with any IPC. We will focus on supporting one form of IPC, either the RPC in zoobar or LCM [2] (Lightweight communication and Marshaling), depending on the difficulty of the task. LCM is a communication module developed at MIT for the Darpa Grand Challenge and allows processes to share data using UDP multicast. Many robotic systems developed at MIT still use LCM, and having a concolic execution system that can test these systems would be of a tremendous value.

We propose to support IPC between processes by encapsulating all the involved processes under the same concolic execution framework. As opposed to taking concrete values as communicated input to processes, we propose to map the symbols across all processes by jointly testing all the involved process together. This approach would perceptually be a large tree where all the branches are parts of different processes. One might wonder that if we link all processes, they might form a cycle, not a tree. It is likely so because processes usually tend to transmit data to each other. In this case, we may be able to employ existing analysis techniques such as abstract interpretation. By linking the variables across processes, we should be able to achieve this.

3 Design and Implementation Details:

As the starting point, we propose to adapt an existing C/C++ based concolic execution framework to support several processes at once. This would require doing a survey of the existing frameworks and choosing the most appropriate one. We expect to modify an existing framework to either share symbols across multiple instances of itself, or run several processes within one instance. Thorough investigation into the existing frameworks would be necessary for the exact implementation details. Then, we plan to apply the modified concolic execution on either the zoobar application or a real robotic system at MIT, e.g. Darpa Urban Challenge, UAV navigation platforms, Darpa Robot Challenge, etc, to report the code coverage and any possible bugs we may find.

References

[1] Cristian Cadar, Daniel Dunbar, and Dawson Engler. Klee: Unassisted and automatic generation of high-coverage tests for complex systems programs. In *Proceedings of the 8th USENIX Conference on Operating Systems Design and Implementation*, OSDI'08, pages 209–224, 2008.

[2] A.S. Huang, E. Olson, and D.C. Moore. Lcm: Lightweight communications and marshalling. In *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, pages 4057–4062, Oct 2010.