CoDeTest: Comprehension, Detection and Testing via Symbolic Execution

Principal Investigator: Abhik Roychoudhury, NUS

Co-Investigator: Kenneth Cheong, DSO Labs

1 Post-doctoral research fellow, and 1 Research assistant position is being announced for this project. The positions can start anytime from June 2013. The positions are for up to 2 years. Interested applicants can contact the Principal Investigator or Co-Investigator for more queries.

Short description:

Software is the heart of our societal infra-structure and software reliability is thus of paramount importance. In this project, we aim to build integrated analysis engines for C, C++ binaries. Our work-plan consists of joint research between NUS School of Computing and DSO Labs in building an integrated analysis tool which encompasses both program dependency analysis and symbolic execution. We also discuss specific usages of our analysis engine – which we plan as part of this project. These are:

[Supporting strong isolation of critical software] Current generation critical mobile apps (such as those handling sensitive data) are physically coexisting with non-critical data and apps in smart-phones. To prevent leaking of sensitive data, strong logical isolation guarantees are needed. In the absence of suitable hardware support to do so, the isolation techniques may be implemented in software and can emulate the hardware support. Our binary analysis can help guarantee/establish the requisite strong isolation.

[“Understanding Obfuscated Software”] Malicious software often comes in obfuscated forms, and the malicious behavior may only be triggered for certain inputs. Symbolic execution based test generation can help in exploring the entire input space and finding the hidden malicious behavior in obfuscated code. However, even after finding the (manifestation of) malicious behavior, it might not be easy to understand it. Dependency analysis tools can help in this regard.

[Witnessing execution of “problematic locations”] If the developer can identify a problematic location in a program, our tool-chain can be used to find an input which can witness the execution of the problematic location. Synthesizing such an input requires the power of both dependency analysis and symbolic execution.

[“Continuous Software Testing”] As a developer makes changes in software, vulnerabilities get introduced and existing functionality can break (regressions). One way to detect such regressions early, is to have a change analyzer run in the background (as part of the programming environment).