Abstract

MutaGen: Fuzz-testing using dynamic data-flow assisted program mutation

Ulf Kargén
IDA, Linköping University
ulf.kargen@liu.se
Supervisor: Nahid Shahmehri

Fuzzing, or fuzz-testing, has become one of the most important security testing techniques employed today, and is a mandatory part of the quality and security assurance processes at many software companies. Being an automated black-box test-generation technique, fuzzing has proven highly effective at uncovering software defects due to unsound assumptions or oversights among developers. Consequently, innumerable exploitable security bugs in popular software have been discovered using fuzzing.

Fuzzing comes traditionally in two flavors: mutational and generational. The former is the simpler of the two, and simply entails doing random mutations (e.g. simple bit flipping) on a well-formed initial program input. While very simple to implement and requiring little human effort, mutational fuzzing often brakes the structure of inputs, resulting in early input rejection, and therefore very poor code coverage. Mutational fuzzing, however, does not require access to source code, or even knowledge of input formats. This has made the technique popular among independent security analysts auditing closed-source software for bugs, and contributed significantly to its wide-spread use. Generational fuzzing instead makes use of a manually-written input-format specification (e.g. a grammar) to generate semi-valid inputs. While code coverage is typically significantly better, the substantial human effort required for crafting format specifications is an impediment to the use of generational fuzzing. Generational fuzzing is naturally also limited to programs with well-documented input formats.

Our proposed method, which was presented at FSE'15, aims to marry the higher code-coverage of generational fuzzing with the low human effort and input-format agnosticism of mutational fuzzing. Our method is based on the observation that, even for closed-source programs with proprietary input formats, an implementation that can generate well-formed inputs to the program is typically available. By systematically mutating the program code of such generating programs, we leverage information about the input format encoded in the generating program to produce high-coverage test inputs, capable of reaching deep states in the program under test. Our method works entirely at the machine-code level, enabling use-cases similar to traditional mutational fuzzing.

In our talk, we will briefly outline the key elements of our method, and also present some of the results from our empirical evaluation, which indicates a significantly higher code coverage and bug-finding capability compared to ordinary mutational fuzzing.