# Random Testing for Security: Blackbox vs. Whitebox Fuzzing

## Invited Talk

Patrice Godefroid

Microsoft Research
pg@microsoft.com

## ABSTRACT

*Fuzz testing* is an effective technique for finding security vulnerabilities in software. Fuzz testing is a form of *blackbox random* testing which randomly mutates well-formed inputs and tests the program on the resulting data. In some cases, *grammars* are used to randomly generate the well-formed inputs. This also allows the tester to encode application-specific knowledge (such as corner cases of particular interest) as part of the grammar, and to specify test heuristics by assigning probabilistic weights to production rules. Although fuzz testing can be remarkably effective, the limitations of blackbox random testing are well-known. For instance, the `then` branch of the conditional statement "`if (x==10) then`" has only one in $2^{32}$ chances of being exercised if `x` is a randomly chosen 32-bit input value. This intuitively explains why random testing usually provides low code coverage.

Recently, we have proposed an alternative approach of *whitebox fuzz testing* [4], building upon recent advances in dynamic symbolic execution and test generation [2]. Starting with a well-formed input, our approach symbolically executes the program dynamically and gathers constraints on inputs from conditional statements encountered along the way. The collected constraints are then systematically negated and solved with a constraint solver, yielding new inputs that exercise different execution paths in the program. This process is repeated using a novel search algorithm with a coverage-maximizing heuristic designed to find defects as fast as possible in large search spaces. For example, symbolic execution of the above code fragment on the input `x` = `0` generates the constraint `x` $\neq$ `10`. Once this constraint is negated and solved, it yields `x` = `10`, which gives us a new input that causes the program to follow the `then` branch of the given conditional statement.

We have implemented this approach in SAGE (*Scalable, Automated, Guided Execution*), a tool based on x86 instruction-level tracing and emulation for whitebox fuzzing of file-reading Windows applications. While still in an early stage of development and deployment, SAGE has already discovered more than 30 new bugs in large shipped Windows applications including image processors, media players and file decoders. Several of these bugs are potentially exploitable memory access violations.

In this talk, I will briefly review blackbox fuzzing for security testing. Then, I will present an overview of our recent work on whitebox fuzzing [4] (joint work with Michael Y. Levin and David Molnar), with an emphasis on the key algorithms and techniques needed to make this approach effective and scalable (see also [1, 3]).


*RT* '07, November 6, 2007, Atlanta, GA, USA


## Categories and Subject Descriptors

D.2.4 [**Software Engineering**]: Software/Program Verification; D.2.5 [**Software Engineering**]: Testing and Debugging; F.3.1 [**Logics and Meanings of Programs**]: Specifying and Verifying and Reasoning about Programs

## General Terms

Verification, Algorithms, Reliability

## Keywords

Software Testing, Automatic Test Generation, Program Verification, Security

## 1. REFERENCES

[1] P. Godefroid. Compositional Dynamic Test Generation. In *Proceedings of POPL'2007 (34th ACM Symposium on Principles of Programming Languages)*, pages 47–54, Nice, January 2007.

[2] P. Godefroid, N. Klarlund, and K. Sen. DART: Directed Automated Random Testing. In *Proceedings of PLDI'2005 (ACM SIGPLAN 2005 Conference on Programming Language Design and Implementation)*, pages 213–223, Chicago, June 2005.

[3] P. Godefroid, M.Y. Levin, and D. Molnar. Active Property Checking. Technical Report MS-TR-2007-91, Microsoft, July 2007. See `http://research.microsoft.com/users/pg/`.

[4] P. Godefroid, M.Y. Levin, and D. Molnar. Automated Whitebox Fuzz Testing. Technical Report MS-TR-2007-58, Microsoft, May 2007. See `http://research.microsoft.com/users/pg/`.