Chapter 9
CONFU:
Configuration Fuzzing Testing Framework for Software Vulnerability Detection

Huning Dai
Columbia University, USA

Christian Murphy
Columbia University, USA

Gail Kaiser
Columbia University, USA

ABSTRACT

Many software security vulnerabilities only reveal themselves under certain conditions, that is, particular configurations and inputs together with a certain runtime environment. One approach to detecting these vulnerabilities is fuzz testing. However, typical fuzz testing makes no guarantees regarding the syntactic and semantic validity of the input, or of how much of the input space will be explored. To address these problems, the authors present a new testing methodology called Configuration Fuzzing. Configuration Fuzzing is a technique whereby the configuration of the running application is mutated at certain execution points to check for vulnerabilities that only arise in certain conditions. As the application runs in the deployment environment, this testing technique continuously fuzzes the configuration and checks "security invariants" that, if violated, indicate vulnerability. This paper discusses the approach and introduces a prototype framework called ConFu (CONfiguration FUzzing testing framework) for implementation. Additionally, the results of case studies that demonstrate the approach's feasibility are presented along with performance evaluations.

INTRODUCTION

As the Internet has grown in popularity, security testing is undoubtedly becoming a crucial part of the development process for commercial software, especially for server applications. However, it is impossible in terms of time and cost to test all configurations or to simulate all system environments before releasing the software into the field, not to mention the fact that software distributors may later add more configuration options. The configuration of a software system is a set of

DOI: 10.4018/978-1-4666-1580-9.ch009

options that are responsible for a user's preferences and the choice of hardware, functionality, etc. Sophisticated software systems always have a large number of possible configurations, e.g., a recent version of Firefox has more than 2^{30} possible configurations, and testing all of them is infeasible before the release. Fuzz testing as a form of black-box testing was introduced to address this problem (Sutton et al., 2007), and empirical studies (Jurani, 2006) have proven its effectiveness in revealing vulnerabilities in software systems. Yet, typical fuzz testing has been inefficient in two aspects. First, it is poor at exposing certain errors, as most generated inputs fail to satisfy syntactic or semantic constraints and therefore cannot exercise deeper code. Second, given the immensity of the input space, there are no guarantees as to how much of it will be explored (Clarke, 2009).

To address these limitations, this paper presents a new testing methodology called *Configuration Fuzzing*, and a prototype framework called *ConFu* (CONfiguration FUzzing framework). Instead of generating random inputs that may be semantically invalid, ConFu mutates the application configuration in a way that helps valid inputs exercise the deeper components of the software-under-test and check for violations of program-specific "security invariants" (Biskup, 2009). These invariants represent rules that, if broken, indicate the existence of a vulnerability. Examples of security invariants may include: avoiding memory leakage that may lead to denial of service; a user should never gain access to files that do not belong to him; critical data should never be transmitted over the Internet; only certain sequences of function calls should be allowed, etc. ConFu mutates the configuration using the incremental covering array approach (Fouche et al., 2009), therefore guaranteeing considerable coverage of the configuration space in the lifetime of a certain release of the software.

Configuration Fuzzing works as follows: Given an application to test, the testers annotate the variables to be fuzzed in the configuration file and choose the functions to test. If needed, they can write additional surveillance functions for specific security invariants other than the built-in ones provided by our default implementation. The framework then generates the actual code for a fuzzer that mutates the values of the chosen configuration variables, as well as the test functions for each chosen function. Next, the framework creates instrumentation such that whenever a chosen function is called, the corresponding test function is executed in a sandbox with the mutated configuration and the security invariants are checked. Violations of these security invariants are logged and sent back to the developer.

Configuration Fuzzing is based on the observation that most vulnerabilities occur under specific configurations with certain inputs (Ramakrishnan & Sekar, 2002), i.e., an application running with one configuration may prevent the user from doing something bad, while another might not. Configuration Fuzzing occurs within software as it runs in the deployment environment. This allows it to conduct tests in application states and environments that may not have been conceived in the lab. In addition, the effectiveness of ConFu is increased by using real-world user inputs rather than randomly generated ones. However, the fuzzing of the configuration occurs in an isolated "sandbox" that is created as a clone of the original process, so that it does not affect the end user of the program. When a vulnerability is detected, detailed information is collected and sent back to a server for later analysis.

The rest of this paper is organized as follows. The problem statement, and identifies requirements that a solution must meet is formalized. The next section discusses the background, proposes the Configuration Fuzzing approach, and provides the architecture of the framework called ConFu. The results of our case studies and performance evaluation are then examined. Related work is then discussed. The paper ends with limitations and a conclusion.