A Fuzz Testing Framework for Evaluating and Securing Network Applications

Sascha Zeisberger and Barry Irwin
Department of Computer Science and Information Systems
Rhodes University, P.O. Box 94, Grahamstown 6140
Tel: +27 46 6038291, Fax: +27 46 6361915
email: s.zeisberger@live.com, b.irwin@ru.ac.za

Abstract—Research has shown that fuzz-testing is an effective means of increasing the quality and security of software and systems. This project proposes the implementation of a testing framework based on numerous fuzz-testing techniques. The framework will allow a user to detect errors in applications and locate critical areas in the applications that are responsible for the detected errors. The aim is to provide an all-encompassing testing framework that will allow a developer to quickly and effectively deploy fuzz tests on an application and ensure a higher level of quality control before deployment.

Index Terms—Security and Privacy, Software Design, Web & Applications servers

I. INTRODUCTION

Fuzz testing refers to the automation of generating malformed data and sending that data to an application’s input. This is done so that an evaluation of an application’s robustness can be made. Popular inputs to fuzz test include application arguments, files and network communications, although in recent years more unconventional targets, such as the SMS stack on mobile phones, have been targeted [1][2]. For the purpose of this project, focus will be on fuzzing application layer protocols.

Fuzz-testing has been successfully employed in development environments alongside other testing tools and has resulted in the development of more secure applications [3]. Despite this observation, fuzz-testing is rarely used in development environments outside of security research groups [1]. This project explores the implementation of fuzz-testing framework that will aid a developer in locating and fixing errors in an application and allow system builders to evaluate the robustness of vendor applications. The main goal of this framework is to make fuzz-testing more accessible to developers in the quality assurance process.

II. BACKGROUND

Fuzz-testing aims to reveal flaws in input and error handling routines in applications [4]. When fuzz-testing is applied to applications outside of development environments, it can be used to test the quality of an application and aid system builders in evaluating software solutions [5]. This becomes crucial when designing large systems consisting of several applications, such as telephony systems, as the security of the overall system may be comprised by a single weak component.

A. Fuzzer Characteristics

Basic requirements that are common to all fuzzers include a specification of the data that is being fuzzed, a tracking mechanism that allows the fuzzing process to be replayed in the event of an error, and a method of creating and sending malformed data using the definition that describes the data [1][6].

Fuzz-testing tools can be classified depending on a range of characteristics, the most prominent characteristic referring to how the malformed data is selected; either the data is generated or it is mutated [1]. A fuzzer that generates data requires some description of the data that it is fuzzing and generates fuzz test-cases according to the description of the data. Using this technique, an email field could possibly be described as “<string>@<string>.<string>” and the generation algorithm will create samples based on that description. A fuzzer that mutates data takes a valid sample of data, analyses the structure of the data, and then mutates the data using predefined rule sets and mutation algorithms to create fuzz test-cases. Staying with the email example, a mutation algorithm may analyse a valid email address and then perform a series of operations on that string to create fuzzed samples. Operations may include repeating, deleting and inserting characters.

Fuzzers can also be classified depending on whether they are stateless or stateful [1]. A stateless fuzzer is only able to test the first message in a sequence of messages. A stateful fuzzer can test entire sequences, such as when initiating a call using SIP [7]. According to [7], a stateful fuzzer may be more useful when testing complex protocols as it can reveal errors in large sequences, whereas a stateless fuzzer would only test the initial transaction in a sequence.

B. Fuzzing Approaches

The strategy employed when fuzz-testing an application is determined by the level of access to the application. In situations where the source code is available, known as white box testing, the fuzz data can be narrowed down as test-cases, relevant to the source code, can be specified [1]. A simple example of this would be a situation where an application filters and ignores transactions consisting of a series of “A” characters. The fuzzer could be configured to not fuzz using “A” characters, but rather “B” characters. In situations where only the application’s binary is accessible, known as grey box testing, the number of test-cases

The authors would like to acknowledge the financial support of Telkom SA, Stortech, Tellabs, Easittel, Bright Ideas 39 and THRIP through the Telkom Centre of Excellence in the Department of Computer Science at Rhodes University.
increases as the user has no view of the application’s underlying behaviour [1]. In situations where only remote access to the application is possible, known as black box testing, the same issue of the application’s underlying behaviour being unknown exists. Black box testing’s short fall is that advanced tools that require access to the application binary, such as debuggers, cannot be used to monitor an application’s state in the event of an error [1].

C. Shortfalls of fuzzers

As with many testing frameworks, fuzzers will not reveal all the problems within an application [1]. This issue becomes important when logic flaws occur in an application as a result of fuzzing. For example, when fuzzing a system with multiple user levels, the fuzzer is incapable of tracking whether a fuzzed entity’s privileges have become escalated.

In black-box testing environments, where the tester is limited to remote access to the host system, the fuzzer has no method of tracking the status of that remote application. This is usually overcome by having the fuzzing application send a valid test-case to the remote application after a certain number of executed test-cases. This can become costly in high contention networks where bandwidth is limited.

Fuzzing frameworks that do not target a specific protocol often require an intermediate language to describe the protocol being fuzzed [8][9]. This means that a developer, when fuzz-testing, may have to learn an unfamiliar language, adding to the time and cost of conducting fuzz tests.

III. PROPOSED SOLUTION

A fuzzing framework is in the process of being implemented that will abstract the fuzzing process from the user in an attempt to simplify. This is to simplify fuzz testing and create a cost effective solution to performing fuzz tests on applications. The goal is to create a scenario where the user starts the application, starts the fuzz testing process and once testing is complete, is presented with a report that will outline what inputs caused errors and the status of the application at the time of the error.

The fuzzing framework will not be responsible for any fuzzing. Due to the intricacies of fuzzing, it will interact with other well-established fuzzers in a similar fashion as the Radamsa file fuzzing framework, where a single method of interaction is used to manage multiple fuzzers [4]. This allows for a simple approach to managing multiple fuzzers and fuzzing techniques and requires the user to become familiar with a single application, lowering the cost of fuzz-testing.

The design of the framework will be based on the Sulley fuzzing framework and will be modularized into three main components; a protocol builder, diagnosing and debugging facilities, and a compatibility layer.

The protocol builder provides facilities for the user to describe the protocol to be fuzzed. In order to simplify the initial development of the framework, focus will be on accommodating application layer protocols such as SIP, FTP and X170. This will include the structure of the protocol, its fields and the sequences for that particular protocol. It will store this description in an XML-based format for use later in the application. The protocol builder will be complimented with a graphical user interface so as to simplify the user’s interaction with the fuzzer to describing the protocol in a graphical manner. This is to eliminate the need to learn an intermediate language when using generic fuzzers.

Diagnosing and debugging facilities will initially be limited to packet captures and debugging facilities provided by the Sulley fuzzing framework, namely Paimei and PyDbg [8]. This will allow the framework to monitor the application state during testing and can provide detailed reports on the state of the host system at the time of an error [8].

The compatibility layer will handle the communications between the developed framework and the fuzzers. This will include converting the XML-based description of the protocol created by the protocol builder to a recognisable format for the selected fuzzers. The compatibility layer will also be responsible for communicating with and managing the fuzzers. A benefit of using this approach is that 3rd party developers will be responsible for maintaining the fuzzing components of the framework, while the maintenance of the fuzzing framework itself will be isolated to the compatibility layer.

IV. CONCLUSION

Fuzzing is necessary to produce stable production code but becomes a complex process as current solutions are either too complex or do not cover a large enough scope to work effectively. It is for this reason that the implementation of a new fuzzing framework has been proposed. The framework will aim to lower the cost of executing fuzz-tests and will use a variety of fuzz-testing techniques by providing a single front-end to a variety of current fuzz-testing solutions.

V. REFERENCES


Sascha Zeisberger is presently studying towards his MSc in Computer Science at Rhodes University. His research interests include Information Security and mobile development.