Fuzzing, an exploration of Mind, Body, and Argentina

Stephens, Nick
nick.stephens93@gmail.com

Boesen, Stefan
stefan.boesen@gmail.com

David, Max
maxs-email@404

June 26, 2014

Abstract

This paper intents to summarize a quarter of undergraduate research in *fuzzing*. In Section 1 we introduce fuzzing, methodologies, and best practices. Section 2 describes our experiences implementing those techniques on a real world library, the MPD project. Section 3 contains our closing thoughts, including words of advice for those interested.

1 Introduction and Intent

Our learning contract was intended to expand our understanding of fuzzing and how it applies to real world projects. Fuzzing: Brute Force Vulnerability Discovery [cite] was our primary reference, being a well known introductory book on the topic. Conceptually fuzzing is a very simple topic. By feeding unanticipated input into applications that do not properly handle them, unexpected behavior can be introduced into otherwise functioning programs. As an example, a program may attempt to read a number from a network connection. What happens if the program is sent a character instead? Is the character interpreted as a number, or does it just fail?

The goal of fuzzing is typically to identify locations of potential bugs. Typically this entails having a component reading the "state" of some program, whether that's the error log, signals sent, or other behaviors. As the goal is to produce unexpected behavior, it can be difficult to know if you have anticipated all of the possible states. Fuzzing has other challenges as well, such as identifying inputs to target as well as determining what input to generate. Instead of testing every number, perhaps it's more valuable to try extremes and random ones throughout. Strings with unusual characters may be preferred.

A final challenge is that simplistic fuzzing measures the state after each input, when in reality, a system's state is affected by previous commands as well. This means that an environment should be set up where one can return to a previous state quickly (using a local virtual machine with memory snapshotting) can be critical for testing sequences of inputs.

Our goal going into this project was to implement a basic fuzzer in order to discover vulnerabilities in an open source program we had all used, *MPD*.

2 Fuzzing

3 Conclusion

Fuzzing is easy if the codebase is terrible.