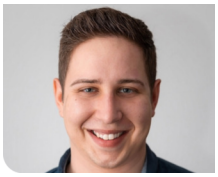
Systematic Identification of Vulnerabilities in C and C++ Source Code through Fuzzing

Advanced Techniques for Efficient Vulnerability Detection

Graduate



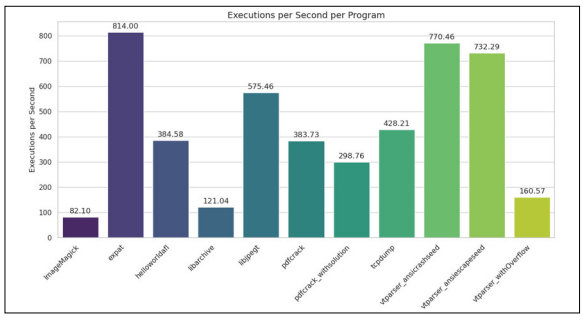
Miles Strässle

Introduction: As software becomes more complex and security issues in applications grow. This research focuses on the use of high-performance fuzzing techniques and also investigates performance parameters for fuzzing in different contexts. Fuzzing is a method of finding software vulnerabilities by injecting random data into programs to reveal and being able to fix potential security flaws. The goal is to use an advanced fuzzing framework to identify vulnerabilities in real-world open-source C and C++ software, thereby improving its robustness and security.

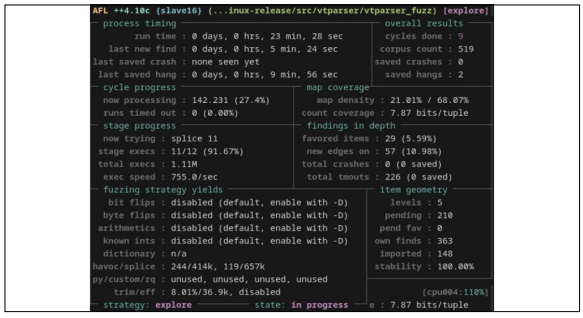
Approach: To find a suitable project for fuzzing, the search targeted software that accepts input from users or external sources, focusing on areas most likely to contain vulnerabilities. A variety of open-source C and C++ projects with significant user interaction components were selected. A fuzzing harness was then created to test these critical areas of the software, utilizing various inputs and seed values. Employing white-box fuzzing, full access to the source code allowed for more informed tests, simplifying bug identification and avoiding reverse engineering as in black-box fuzzing. Once the tests were executed, the resulting bugs and hangs were analyzed to understand their causes and potential security implications. Metrics such as the number of bugs found, the types of vulnerabilities, and the duration of tests were collected to assess the effectiveness of the fuzzing process.

Conclusion: This project used advanced fuzzing techniques to test real-world C and C++ open-source projects. The fuzzing framework successfully reproduced many known security vulnerabilities, proving its effectiveness and reliability. Although no new vulnerabilities were found, the high performance and efficiency of the setup make it suitable for ongoing security testing. The metrics collected - such as the number and types of bugs found and test durations - highlighted areas for improvement. The results demonstrate the robustness of the code in the context of penetration testing and security audits, underscoring the importance of continuous security testing and how fuzzing can enhance software security.

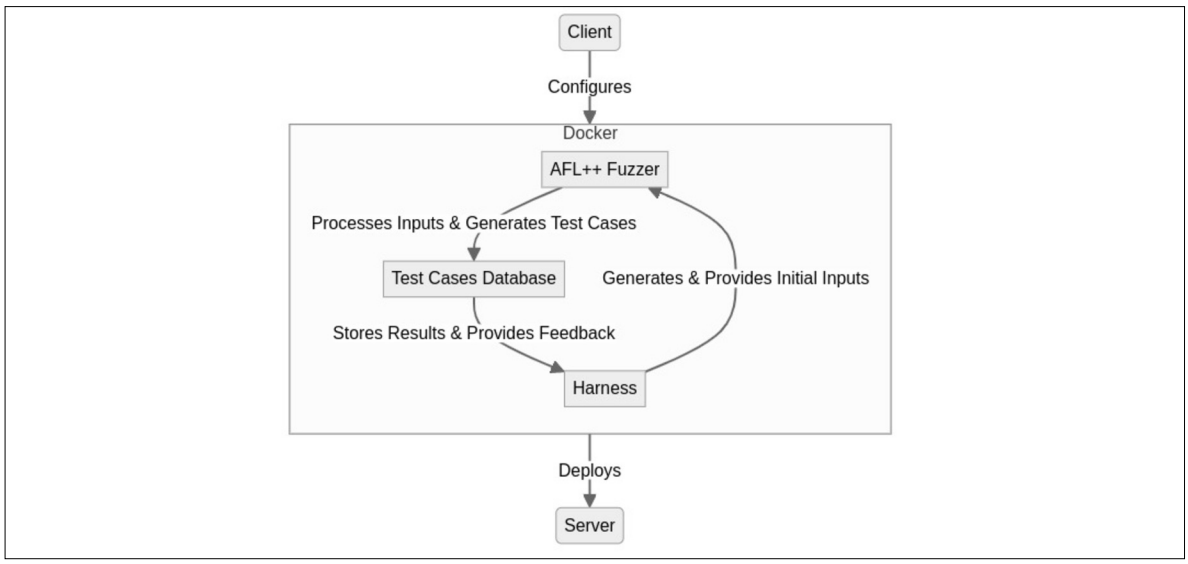
Execution Speed per Fuzzing-Project
Own presentment



Screenshot of AFL++ running on vt-parser.
Own presentment



Overview of the AFL Fuzzing Process
Own presentment



Advisor
Nikolaus Heners

Co-Examiner
Thomas Sutter, ZHAW School of Engineering, Winterthur, ZH

Subject Area
Security, Software