Fuzzer Research

Jack Foley

October l4, 2024

1 Introduction

This document will outline the research undertaken for the 4th Software Development Final Year Project (FYP). This project was proposed by Dr. Chris Meudec and focuses on developing a fuzzer for the C programming language.

2 Fuzzing

Fuzzing is a method of testing software by using broken, random or unusual data as an input into the software which is being tested. The idea of fuzzing is that it will find bugs and other issues, including memory spikes and leaks (temporary denial-of-service), buffer overruns (remote code execution), unhandled exceptions, read access violations (AVs), and thread hangs (permanent denial-of-service). These are issues that traditional software testing methods, such as unit testing, will not find as easily. There are some different types of fuzzing, such as white-box, grey-box and black-box fuzzing [3].

2.1 White-Box Fuzzing

White-Box fuzzing, also known as smart fuzzing, is a technique that is used where the fuzzer is fully aware of the code structure and input variables. white-box fuzzing often leads to discovering bugs more quickly compared to grey-box and black-box fuzzing, but it can also be more computationally expensive as it needs to do an analysis of the codebase before running.

An example of a well known white-box fuzzer is **SAGE** [9], a white-box fuzz testing tool used for finding bugs in security vulnerabilities of software applications. It was designed by Microsoft Research. Unlike traditional black-box fuzzing that generates random inputs without any idea about program structure, SAGE analyses all possible code paths of the program using symbolic execution. Thus, SAGE creates inputs systematically in order to explore a lot of execution paths. This exposes bugs missed by other testing techniques.

A case study done during development of ISA Server 2006 showed that one defect was found per 17 KLOC (thousand lines of code), a similar black-box fuzzer only found 30% of the defects that the white-box fuzzer found [3].

2.2 Black-Box Fuzzing

Black-Box fuzzing is a technique used to test software, analysing the software by sending random data to the software to discover an application's bugs and vulnerabilities. The black-box fuzzer does not have any information about the inner-workings of the software, it only knows the input and output of the software.
It is a sought-after testing technique as it will work in applications regardless of the programming language or the platform that the software is running on [1].

A black-box fuzzer can typically get information about what the input to the fuzz target is through some of the following methods:

- **Protocol Specifications:** If the fuzz target follows well known protocol standards such as HTTP or HTTPS, the fuzzer will create inputs that are based on the HTTP protocol.
- **Sample Inputs (Seed Files):** black-box fuzzers will often start with a set of known valid inputs, also known as seed inputs. These inputs are then manipulated after each subsequent test execution. Some well known fuzzers, such as American Fuzzy Lop (AFL) use this method to improve code coverage during fuzz testing [10].
- **Dynamic Analysis and Learning:** Some sophisticated fuzzers interpret the application's responses to random inputs as a means to infer the structure of the expected inputs. This may include the capability of the fuzzer to learn from how the application was processing previous inputs, adaptively refining its test cases. For example, the Snipuzz framework uses message snippet inference to guide its fuzzing process [11].

2.3 Grey-Box Fuzzing

Grey-Box fuzzing is a well-known and commonly used fuzzing technique that is used for testing software and finding vulnerabilities. Differing from white-box fuzzing, which can suffer from high computational needs since source code analysis is required, grey-box fuzzing is a very good middle-ground between white-box and black-box fuzzing [4].

Grey-Box fuzzing can also receive coverage feedback from the software, which can then be used to more efficiently traverse the software's codebase to find bugs and vulnerabilities [2].

3 Techniques Deep Dive

3.1 Random Input Fuzzing

The simplest implementation of a Fuzzer is a Random Fuzzer. This type of Fuzzer will generate a random string at a fixed or variable length which will then be used as the input for the software which we are testing. The method of random fuzzing is extremely efficient [13], but may struggle at producing inputs that do not cause errors [6].
Examples of random fuzzing would be: **`*&322h2k,b&(Gb2\|q&@ih`**

3.2 Mutation Based Fuzzing

Instead of generating completely random strings, we can use mutation based fuzzing. Most randomly generated inputs are always invalid, which is not ideal. Mutation Fuzzing will take a valid input at first, then with each subsequent execution, it will change, or mutate, the string slightly. This mutation is usually done by modifying one random character in the input. This approach is popular with fuzzing as it may cause the program to crash while only changing the input slightly, which is difficult to achieve with traditional testing [7].

Examples of mutation fuzzing would be:

- Original Input: Hello World
- Mutated Input l: Hello Wzrld
- Mutated Input 2: Helll Wzrld
- Mutated Input 3: H;lll Wzrld
- ...
- Mutated Input N: Fr'l?.tOP4+

3.3 Coverage Guided Fuzzing

Coverage based fuzzing will trace back the code coverage reached by each input fed to a program. The fuzzing engine will then decide which section of the next input should be modified (or mutated) to achieve the maximum amount of code coverage within its next test.

4 Benchmarking Overview

4.1 Competitions

4.1.1 Test-Comp

Test-Comp (TC) is an annual international competition that focuses on determining the technological level of advanced software testing tools. The competition is made to do the following:

- Assess the state of current automated software testing.
- Give recognition to the developers of these tools by providing a platform for them.
- Create a thorough set of benchmarking standards for software testing.

Users are given a set of standardised benchmark programs that they can use during a training phase to prepare their tools. In the subsequent evaluation phase, these tools are executed on specific test tasks to measure their effectiveness in bug detection, code coverage, and runtime performance. The competition is organised and presented by the Test-Comp competition chair [12].

4.1.2 SV-Comp

SV-Comp (SVC) is a software verification benchmark website. It runs an annual competition to test various different software verification tools which can be used in the software development lifecycle. It is mostly used to prove the correctness of a software verification tool while following formal specifications, but it seems that there are some fuzzers used in the competition. SVC does not seem like it will be a good candidate for testing the fuzzer as it is mostly used for the testing of verification tools, not fuzzers. A better alternative is Test-Comp, a software testing competition that is run by the same people who run SVC, but has a higher focus on software testing tools, including fuzzers, rather than software verification tools.

5 Static Code Analysis

5.1 Flex

Flex [14] (Fast lexical analyser generator) is an open-source alternative to lex, another lexical analyser. Its sole purpose is to generate lexical analysers that can be used to

find specific symbols in large files as quickly as possible. Since we are working with defined problem files from an official benchmark, we can look for certain variable names that *should* show up somewhere in the problem files. Using the lexer, we can find how these symbols and/or variables are being used in the problem file and then using that information we can create the correct and expected inputs from the fuzzer.

5 References

1. Aseel Alsaedi, Abeer Alhuzali, and Omaimah Bamasag. “Effective and scalable black-box fuzzing approach for modern web applications”. In: Journal of King Saud University - Computer and Information Sciences 34.10, Part B (2022), pp. 10068–10078. issn: 1319-1578. doi: https://doi.org/10.1016/j.jksuci.2022.10.006.

2. Blackwell, D. and Clark, D. (2024). PrescientFuzz: A more effective exploration approach for grey-box fuzzing. [online] arXiv.org. Available at: https://arxiv.org/abs/2404.18887 [Accessed 28 September. 2024].

3. J. Neystadt. Automated penetration testing with white-box fuzzing. Microsoft Learn. Available at: https://learn.microsoft.com/en-us/previous-versions/software-testing/cc162782(v=msdn.10)?redirectedfrom=MSDN (Accessed: 28 September 2024). 2009.

4. Van-Thuan Pham et al. “Smart Greybox Fuzzing”. In: IEEE Trans-actions on Software Engineering 47.9 (2021), pp. 1980–1997. doi:10.1109/TSE.2019.2941681.

5. Wikipedia contributors. Fuzzing — Wikipedia, The Free Encyclopedia. [Online;accessed 6-October-2024]. 2024. Url: https://en.wikipedia.org/w/index.php?title=Fuzzing&oldid=1249540069.

6. Andreas Zeller et al. “Fuzzing: Breaking Things with Random Inputs”. In: The Fuzzing Book. Retrieved 2024-06-29 17:55:20+02:00. CISPA Helmholtz Center for Information Security, 2024. url: https://www.fuzzingbook.org/html/Fuzzer.html.

7. Andreas Zeller et al. “Mutation-Based Fuzzing”. In: The FuzzingBook. Retrieved 2024-06-29 18:18:06+01:00. CISPA Helmholtz Center for Information Security, 2023. Url:https://www.fuzzingbook.org/html/MutationFuzzer.html.

8. ClusterFuzz. (2024). Coverage guided vs blackbox fuzzing. [online] Available at: https://google.github.io/clusterfuzz/reference/coverage-guided-vs-blackbox/ [Accessed 20 Oct. 2024].

9. queue.acm.org. (n.d.). SAGE: Whitebox Fuzzing for Security Testing - ACM Queue. [online] Available at: https://queue.acm.org/detail.cfm?id=2094081. [Accessed 4 Nov. 2024].

10. Wikipedia Contributors (2024). American Fuzzy Lop (software). Wikipedia. Available at: https://en.wikipedia.org/wiki/American_Fuzzy_Lop_%28software%29 [Accessed 4 Nov. 2024].

11. Feng, X., Sun, R., Zhu, X., Xue, M., Wen, S., Liu, D., Nepal, S. and Xiang, Y. (2021). Snipuzz: Black-box Fuzzing of IoT Firmware via Message Snippet Inference. [online] arXiv.org. Available at: https://arxiv.org/abs/2105.05445 [Accessed 4 Nov. 2024].

12. Sosy-lab.org. (2022). Test-Comp 2022 - 4th International Competition on Software Testing. [online] Available at: https://test-comp.sosy-lab.org/2022/rules.php [Accessed 4 Nov. 2024].

13. Gopinath, R. and Zeller, A. (2019). Building Fast Fuzzers. [online] arXiv.org. Available at: https://arxiv.org/abs/1911.07707 [Accessed 8 Nov. 2024].

14. Wikipedia Contributors (2024). Flex (lexical analyser generator). Wikipedia [Accessed 1 Dec. 2024].