Title: Design and Development of a Custom Browser Fuzzing Tool for Detecting Security Vulnerabilities

Name: Dhiaeddine Kraini
Student Number: 19185073

Supervisor: Hong Zhu

Table of Contents

1. **Introduction**
   1.1 Background
   1.2 Aim
   1.3 Objectives
   1.4 Product Overview
   1.4.1 Scope
   1.4.2 Audience
2. **Background Review**
   2.1 Existing Approaches
   2.2 Related Literature
3. **Methodology**
   3.1 Approach
   3.2 Technology
   3.3 Version Management Plan
4. **Project Management**
   4.1 Activities
   4.2 Schedule
   4.3 Data Management Plan
   4.4 Deliverables
5. **Bibliography**

1. Introduction

1.1 Background

Web browsers have evolved into crucial tools for modern computing, serving as the principal interface through which people access the internet. Browsers accept a variety of untrusted inputs, including HTML, JavaScript, and CSS, all of which are required for online content rendering and execution. However, the complex nature of these inputs makes browsers a tempting target for attackers. Exploiting vulnerabilities in browser components, such as JavaScript engines, HTML parsers, and DOM rendering systems, can result in serious security breaches including data theft, system compromise, and remote code execution [1].

Fuzzing is an effective approach for identifying vulnerabilities in software systems by automatically injecting random inputs. Browser fuzzing has been shown to be extremely successful in detecting weaknesses in browser security by imitating unexpected or malicious input behaviour [2]. Tools like Domato, a DOM fuzzing tool created by Google Project Zero, have helped to find browser vulnerabilities [3]. Existing fuzzing tools, however, have limitations, notably in terms of producing complicated, realistic inputs and detecting minor logic mistakes or memory corruption problems.

The idea behind this project is to create a custom browser fuzzing tool from scratch in order to overcome these constraints. With the goal to uncover vulnerabilities that existing tools might overlook, this tool will automate the creation and altering of test cases across browser components with a focus on increasing input complexity and scaling testing procedures.

1.2 Aim

The primary aim of this project is to design and develop a **custom browser fuzzing tool** that automatically generates, mutates, and executes test cases in major web browsers, with the goal of uncovering security vulnerabilities in JavaScript engines, HTML parsers, and DOM rendering systems.

1.3 Objectives

This project's main goal is to create a custom browser fuzzing tool from the ground up that can automatically create, modify, and run test cases in order to find security flaws in widely used web browsers like Chrome and Firefox. By concentrating on input mutation techniques, this project aims to improve current tools through enabling the fuzzer to produce realistic and complicated inputs that can reveal vulnerabilities in browser components such as DOM rendering systems, HTML parsers, and JavaScript engines.

To further ensure scalability and efficiency in the testing process, the fuzzer will also include an execution engine that automates the process of running these inputs within headless browser setups. A crash detection mechanism will also be incorporated into the fuzzer to track browser activity during testing and log any unusual behaviours or crashes for additional analysis. Lastly, a thorough evaluation of the tool's efficacy in revealing previously undiscovered vulnerabilities will be conducted by testing it on a variety of widely used browsers.

1.4 Product Overview

1.4.1 Scope

The scope of this project is to provide a browser fuzzing tool that will generate and alter test inputs for key browser components automatically. Targeting critical components like HTML/CSS parsers, DOM rendering systems, and JavaScript engines is how the fuzzer will be created. The tool will have a number of important capabilities, including the ability to automatically generate a variety of faulty inputs to mimic real-world attack vectors and run these test cases in headless browser environments, including Firefox and Chrome, to enable large-scale, automated testing. The fuzzer will also include crash detection techniques to keep monitoring out for segmentation faults, memory corruption, and other kinds of browser malfunctions. These crashes and errors will be recorded for further examination and analysis, allowing security researchers or developers to identify and address the underlying causes of the vulnerabilities.

1.4.2 Audience

The tool will primarily benefit:

- **Security Researchers**: To assist in discovering new vulnerabilities in browsers and improve browser security.
- **Browser Developers**: To integrate the fuzzer into their development pipelines, ensuring the robustness and security of browser components.
- **Academic Researchers**: To provide a foundation for further research into fuzzing techniques and browser security.

2. Background Review

2.1 Existing Approaches

A number of fuzzing tools have been created to find vulnerabilities in online browsers. One such tool is Domato, a DOM fuzzing tool created by Google Project Zero, which tests browser components by producing random HTML, CSS, and JavaScript inputs [3]. Although Domato has been successful in identifying vulnerabilities, it has difficulty producing extremely complicated inputs that reveal more serious problems with browser components such as the DOM renderer and JavaScript engine.

IFuzzer is another popular tool that tests rendering engines and browser components for vulnerabilities in the browser by producing invalid HTML [4]. Although it is useful for finding rendering flaws, it only tests the HTML parsing and rendering features of browsers, neglecting other parts such as JavaScript engines.

FuzzBench, an open-source benchmarking platform, provides a standardised environment for evaluating various fuzzers, including ones that target browsers [5]. FuzzBench measures fuzzer performance and provides insights into their efficiency in detecting flaws across various software components. However, it focusses on benchmarking rather than providing solutions for sophisticated input generation that are specific to browser security.

These tools highlight the need for a robust fuzzer capable of producing more complicated inputs and thoroughly testing many browser components, ranging from JavaScript engines to HTML parsers and DOM renderers.

2.2 Related Literature

The idea of symbolic execution for methodically investigating program pathways was first proposed by Godefroid et al. in their seminal work on whitebox fuzzing, which greatly increased test coverage [2]. However, whitebox fuzzing's complexity and resource demands make it less practicable for big systems like browsers, where blackbox fuzzing and mutation-based approaches are more routinely deployed.

Klees et al. undertook a thorough study of fuzzing strategies, revealing that while random input generation is helpful for finding numerous problems, mutation-based fuzzing is more efficient at detecting complicated, subtle vulnerabilities such as use-after-free issues [6]. This facilitates the employment of mutation-based techniques, which this project will utilise to create and enhance browser test inputs.

Furthermore, Zalewski's AFL developed coverage-guided fuzzing, which improves input generation by focussing on inputs that take novel code routes [7]. This project will use similar methodologies, combining mutation-based fuzzing with coverage feedback, to improve test case production and vulnerability detection across several browser components.

3. Methodology

3.1 Approach

The development of the fuzzing tool will follow an Agile methodology, allowing for incremental enhancements and regular testing. The project will begin with a research phase in which existing fuzzing tools such as Domato and Fuzzilli will be evaluated to determine their strengths and weaknesses. This research will help to shape the design of the new tool, specifically the input generation and mutation algorithms, as well as crash detection systems.

After obtaining requirements through research, the development process will be separated into sprints.

Sprint 1 focusses on building the input generation module, which employs mutation techniques to produce test cases for critical browser components (JavaScript engine, HTML/CSS parsers).
Sprint 2: Creates the execution engine, which allows the tool to perform test cases in headless browsers (Chrome and Firefox).
Sprint 3: Utilising AddressSanitizer, integrate the crash detection system to keep an eye out for problems like memory corruption and crashes.
Sprint 4: Extensive testing and assessment are carried out, and the efficacy of the fuzzer is analysed based on code coverage, crash frequency, and the quantity of vulnerabilities found. The tool will also be improved throughout this phase in response to test findings. Testing and tool refinement based on test findings will also be part of this phase.

Future modifications and enhancements to the project will be guided by ongoing input from the research phase and test outcomes. An end-of-phase report including the methodology, test results, vulnerabilities found, and tool performance evaluation will be produced at the end of the development phase.

3.2 Technology

Python will be used for input creation and modification because of its strong support for automation and testing. Headless Test cases will be conducted using Chrome and Firefox, allowing for effective automated testing without a graphical interface. AddressSanitizer (ASan) will be used to identify memory issues and crashes, while AFL or libFuzzer will be used for coverage-guided fuzzing, which ensures effective input creation based on code coverage.

3.3 Version Management Plan

The repository will be hosted on GitHub, and Git will be used for version control. The repository will use a branching method, with the main branch housing stable releases and feature-specific branches for ongoing development work. Regular commits and precise commit messages will guarantee that changes are tracked accurately.

To manage data, all test results, crash logs, and reports will be saved in a shared OneDrive folder. This guarantees that all project files are securely accessed and backed up. The final report, which summarises the project's outcomes, methods, and evaluation results, will also be saved and shared on OneDrive.

4.1 Activities

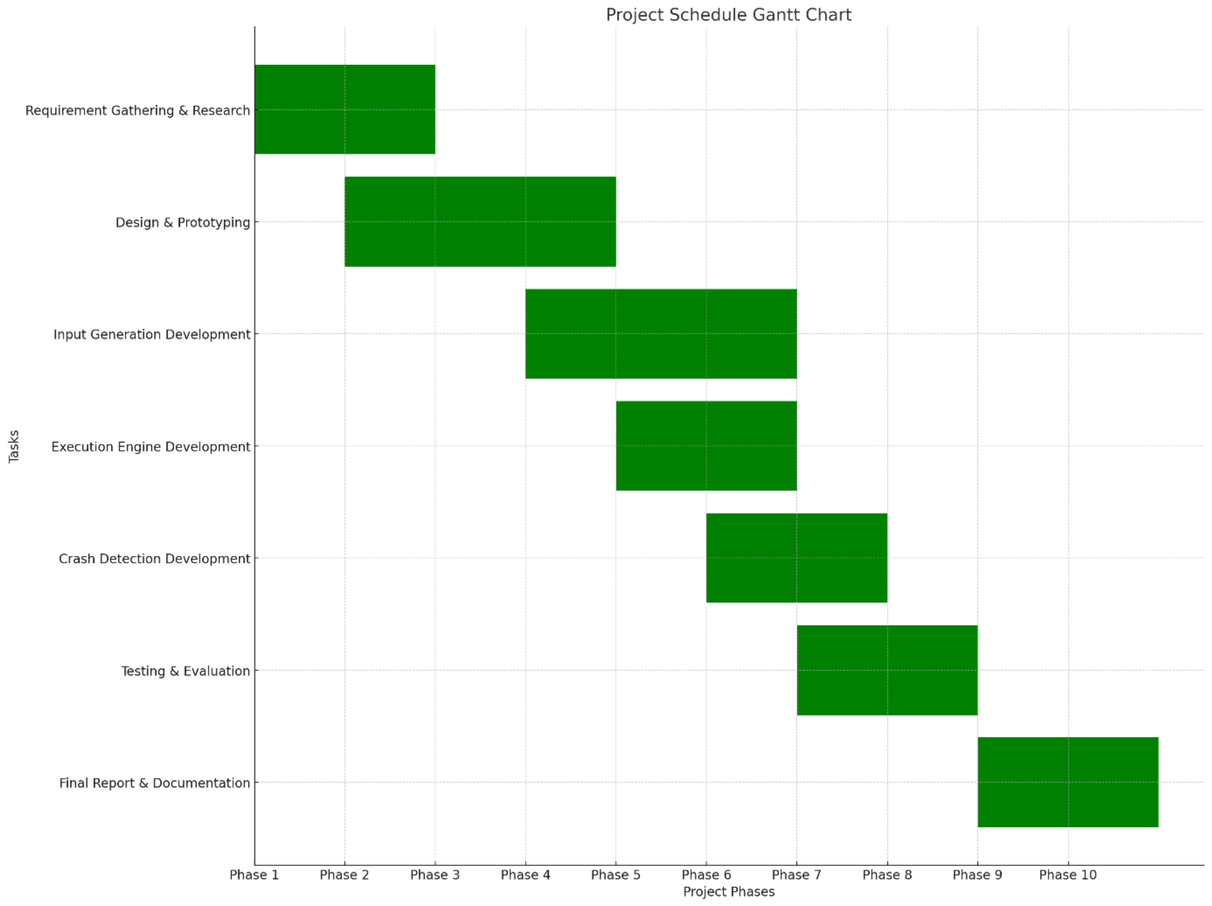
The project will begin with a research phase aiming at studying existing fuzzing tools like Domato and Fuzzilli. With a focus on input generation, mutation strategies, and crash detection mechanisms, this research will lead the definition of the new fuzzing tool's essential features. After that, the architecture of the fuzzer will be structured during the design phase, and prototypes will be created to verify initial concepts.

Once the design is complete, the implementation phase will begin, during which the input generating module, execution engine, and crash detection system will be constructed. The input creation module will employ mutation techniques to generate a wide range of test cases for a variety of browser components, including JavaScript engines and HTML parsers. The execution engine will ensure that test cases run effectively in headless browsers. The crash detection system will check for memory corruption and other irregularities, logging any relevant information.

Following development, the testing phase will analyse the tool's performance by running test cases on several browsers to determine its capacity to find vulnerabilities. Key indicators like as crash frequency, code coverage, and found vulnerabilities will drive further improvement.

Finally, the project will be completed with the gathering of a detailed final report and documentation that summarises the project's objectives, development process, and testing results.

4.2 Schedule



4.3 Data Management Plan

Data generated throughout the project will be securely managed using OneDrive. The folder structure will be organized as follows:

Project Logs: Weekly progress reports, notes, and decisions made during development.
Test Results: Logs and reports from each test run, including crash data, code coverage statistics, and test case results.
Literature and Research Materials: PDFs of research papers, documentation of existing tools, and notes from the research phase.
Final Deliverables: The final report, source code, and user documentation will be backed up and shared here.

4.4 Deliverables

The project will produce the following key deliverables:

Fuzzing Tool: The completed browser fuzzer, capable of generating, mutating, and executing test cases across multiple browsers to identify vulnerabilities.
Documentation: A detailed user manual for setting up and running the fuzzer, along with developer documentation explaining the tool's architecture and components.
Test Results: Logs of crash reports, code coverage data, and other performance metrics generated during testing.
Final Report: A comprehensive report summarizing the project's objectives, development process, testing results, and an analysis of vulnerabilities discovered.

Bibliography

1. J. Fonseca, M. Vieira, and H. Madeira, "Testing and comparing web vulnerability scanning tools for SQL injection and XSS attacks," in *Proceedings of the 13th Pacific Rim International Symposium on Dependable Computing (PRDC 2007)*, Melbourne, Australia, 2007, pp. 365-372.
2. P. Godefroid, M. Y. Levin, and D. Molnar, "Automated whitebox fuzz testing," *Communications of the ACM*, vol. 51, no. 6, pp. 107-113, Jun. 2008.
3. Google Project Zero, "Domato: DOM Fuzzing Tool," GitHub. [Online]. Available: https://github.com/googleprojectzero/domato. [Accessed: Oct. 15, 2024].
4. S. Veggalam, S. Rawat, I. Haller, and H. Bos, "IFuzzer: An Evolutionary Interpreter Fuzzer Using Genetic Programming," in Computer Security – ESORICS 2016, I. Askoxylakis, S. Ioannidis, S. Katsikas, and C. Meadows, Eds. Cham, Switzerland: Springer, 2016, pp. 527-548. [Online]. Available: https://doi.org/10.1007/978-3-319-45744-4_29.
5. J. Metzman, L. Szekeres, L. Simon, R. Sprabery, and A. Arya, "FuzzBench: An open fuzzer benchmarking platform and service," in Proceedings of the 30th USENIX Security Symposium (USENIX Security '21), 2021, pp. 1393-1403. doi: 10.1145/3468264.3473932.
6. G. Klees et al., "Evaluating fuzz testing," in Proceedings of the IEEE Symposium on Security and Privacy, San Francisco, CA, USA, May 2018, pp. 530-545.
7. M. Zalewski, American Fuzzy Lop: A Security-Oriented Fuzzer, 1st ed., Google Inc., 2014.