

**MULTI-FUZZER TECHNIQUES FOR AUTOMATED  
VULNERABILITIES ASSESSMENTS**

By

Choy Ein Jun

A REPORT

SUBMITTED TO

Universiti Tunku Abdul Rahman

in partial fulfillment of the requirements

for the degree of

**BACHELOR OF COMPUTER SCIENCE (HONOURS)**

Faculty of Information and Communication Technology  
(Kampar Campus)

JANUARY 2025

## **COPYRIGHT STATEMENT**

© 2025 Choy Ein Jun. All rights reserved.

This Final Year Project report is submitted in partial fulfillment of the requirements for the degree of Bachelor of Computer Science (Honours) at Universiti Tunku Abdul Rahman (UTAR). This Final Year Project report represents the work of the author, except where due acknowledgment has been made in the text. No part of this Final Year Project report may be reproduced, stored, or transmitted in any form or by any means, whether electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of the author or UTAR, in accordance with UTAR's Intellectual Property Policy.

## **ACKNOWLEDGEMENTS**

I would like to extend my deepest gratitude to my supervisor, Dr. Aun Yichiet, for his invaluable guidance, continuous support, and insightful feedback throughout this project on multi-fuzzer techniques for automated vulnerabilities assessments. His expertise and encouragement have been instrumental in the successful completion of this research.

Finally, I would like to thank my parents and family for their constant love, support, and encouragement throughout my academic journey. Their belief in me has been a great source of motivation.

## **ABSTRACT**

In the contemporary cybersecurity landscape, effectively safeguarding software from vulnerabilities is critically important. This project introduces an innovative approach to automated vulnerability assessment through a sophisticated multi-fuzzer system designed to enhance the security of at-risk software applications. The primary objective is to provide an efficient and user-friendly solution for identifying and analyzing security vulnerabilities via a dynamic front-end chatbot interface. Users can seamlessly upload their software applications, which are subsequently subjected to a series of diverse fuzzing tools within an automated framework. The system employs a range of fuzzing tools, such as AFL++ and Honggfuzz, ensuring a comprehensive and systematic evaluation of software interfaces and their responses to various potential threats. By automating the fuzzing process, this project facilitates a more efficient and thorough assessment of security weaknesses than traditional manual testing methods. The automated framework generates detailed CVEs on discovered vulnerabilities and potential exploitation scenarios, significantly enhancing the security posture of the evaluated applications. The results of this project demonstrate the system's capability to automatically detect and document vulnerabilities across different software environments, providing a comparative analysis of the effectiveness and limitations of various fuzzing techniques. This analysis offers valuable insights into the roles these techniques play in software security, highlighting the importance of using a multi-fuzzer approach to achieve a more resilient vulnerability assessment. Ultimately, this project underscores the critical role of automation in vulnerability assessment and reinforces the value of employing diverse fuzzing methods as essential tools in advancing cybersecurity practices. The findings contribute to the development of more effective security measures and serve as a foundational resource for improving software security in future applications.

Area of Study (Minimum 1 and Maximum 2): Cybersecurity

Keywords (Minimum 5 and Maximum 10): Fuzzing, Vulnerability Assessment, Automated Security Testing, Program Analysis, Bug Detection, Vulnerability Mitigation



# TABLE OF CONTENTS

<b>TITLE PAGE</b>	<b>i</b>
<b>COPYRIGHT STATEMENT</b>	<b>ii</b>
<b>ACKNOWLEDGEMENTS</b>	<b>iii</b>
<b>ABSTRACT</b>	<b>iv</b>
<b>TABLE OF CONTENTS</b>	<b>v</b>
<b>LIST OF FIGURES</b>	<b>ix</b>
<b>LIST OF TABLES</b>	<b>xiv</b>
<b>LIST OF ABBREVIATIONS</b>	<b>xv</b>
<b>CHAPTER 1 INTRODUCTION</b>	<b>1</b>
1.1 Problem Statement and Motivation	1
1.2 Objectives	2
1.3 Project Scope	3
1.4 Contributions	5
1.5 Report Organization	6
<b>CHAPTER 2 LITERATURE REVIEW</b>	<b>7</b>
2.1 Previous Works on Vulnerabilities Assessment	7
2.2 Fuzzing	15
2.2.1 Introduction to Fuzzing	15
2.2.2 The Synergy of Fuzzing, Symbolic, and Concolic Execution in Software Testing	17
2.2.3 Fuzzing Techniques for Vulnerabilities Detection	22
<b>CHAPTER 3 SYSTEM METHODOLOGY/APPROACH</b>	<b>38</b>
3.1 Conceptual Design	38
3.1.1 Use Case Diagram and Descriptions	38
3.1.2 Activity Diagram	42
3.1.3 Data Flow Diagram (DFD)	45
3.2 Implementation Design	47

3.2.1	System Architecture Diagram	47
3.2.2	Flowchart	49
3.2.3	Internal Fuzzing Logic	53
3.2.4	Fuzzer Architecture	56
3.2.5	Detailed Fuzzing Architecture in FOT	58
<b>CHAPTER 4</b>	<b>SYSTEM DESIGN</b>	<b>59</b>
4.1	System Block Diagram	59
4.2	System Components Specifications	66
4.3	LLM Architecture and CVE Generation Design	69
4.3.1	LLM architecture diagram	69
4.3.2	Transformer Architecture	70
4.3.3	Writing CVEs Based on Crash Reports Using LLMs	71
4.4	System Component Interaction & Data Operations	80
4.4.1	User Interface to Fuzzing Dispatcher	80
4.4.2	Fuzzing Dispatcher to Fuzzing Engines	81
4.4.3	Fuzzing Engines to Result Aggregator	81
4.4.4	JSON Report to LLM Engine	82
4.4.5	Interface Display and Mitigation Execution	82
4.4.6	Summary of Data Operations	83
<b>CHAPTER 5</b>	<b>SYSTEM IMPLEMENTATION</b>	<b>84</b>
5.1	Hardware Setup	84
5.2	Software Setup	85
5.2.1	Software	85
5.2.2	Configuration and Environment Setup	85
5.3	Setting and Configuration	88
5.3.1	Environment Variables	88
5.3.2	Directory Structure Integration	88
5.3.3	Bash Script Configuration	95
5.3.4	PHP & LLM Integration	99
5.3.5	System Resource Tuning	102

5.4	System Operation	103
5.4.1	User Login and Registration	103
5.4.2	Uploading a Binary	106
5.4.3	Fuzzing Begins	108
5.4.4	Crash Detection and Report Combination	109
5.4.5	CVE Report Generation with Mitigation Suggestion	110
5.4.6	Mitigation Action Feedback	114
5.4.7	Before/After Effect Visualization	115
5.4.8	Further Questions Regarding the Uploaded Binary	118
5.4.9	Admin Page with Authentication	119
5.4.10	Account Settings Page	121
5.4.11	Logout Functionality	124
5.5	Implementation Issues and Challenges	125
5.5.1	Fuzzer Not Crashing Consistently	125
5.5.2	Integration of Bash Scripts with PHP Frontend	125
5.5.3	Handling Different Types of Vulnerabilities	126
5.5.4	Difficulty in Implementing Two-Factor Authentication	127
5.5.5	File Permissions and Sudo Requirement	127
5.5.6	Inconsistent Responses for User Queries	128
5.6	Concluding Remark	129
<b>CHAPTER 6</b>	<b>SYSTEM EVALUATION AND DISCUSSION</b>	<b>132</b>
6.1	System Testing and Performance Metrics	132
6.2	Testing Setup and Result	141
6.3	Project Challenges	160
6.4	Objectives Evaluation	163
6.5	Concluding Remark	167
<b>CHAPTER 7</b>	<b>CONCLUSION AND RECOMMENDATION</b>	<b>170</b>
7.1	Conclusion	170
7.2	Recommendation	171

<b>REFERENCES</b>	175
<b>APPENDIX</b>	
A.1 Poster	A-1
A.2 Coding Work	A-2

## LIST OF FIGURES

<b>Figure Number</b>	<b>Title</b>	<b>Page</b>
Figure 2.1.1	Process of Penetration Testing	8
Figure 2.1.2	How does a vulnerability scanner work?	8
Figure 2.1.3	Various steps in the CI/CD pipeline	9
Figure 2.2.1	Overview of Scalable Fuzzing Infrastructure	16
Figure 2.2.2	Overall Process of Fuzzing	16
Figure 2.2.3	Test-Input Generation of Symbolic Execution Algorithm	18
Figure 2.2.4	Concolic Execution Exploration in Function <code>h(int x, int y)</code>	19
Figure 2.2.5	Hybrid Fuzzing Technique's High-Level Architecture	19
Figure 2.2.6	Selective Hybrid Fuzzing Approach (CBS and PSC algorithms)	20
Figure 2.2.7	Conventional Random Mutation Scheme Algorithm	22
Figure 2.2.8	General Workflow of Network Protocol Fuzzing Techniques	24
Figure 2.2.9	Model-based Whitebox Fuzzing Algorithm	25
Figure 2.2.10	Coverage-based Greybox Fuzzing Algorithm	25
Figure 2.2.11	"then" Branch of Conditional Statement	26
Figure 2.2.12	Algorithm: Merging two DOM trees in FREEDOM	27
Figure 2.2.13	Coverage-guided Fuzzing Overview	30
Figure 2.2.14	Algorithm for Coverage-Guided Fuzzing	30
Figure 2.2.15	Algorithm for Fuzzing-Based Grammar Inference	31
Figure 2.2.16	Integer Grammar Example	32
Figure 3.1.1	Use Case Diagram of Multi-Fuzzer Platform for Automated Vulnerability Assessments	38
Figure 3.1.2	Activity Diagram of Multi-Fuzzer Platform for Automated Vulnerability Assessments	42
Figure 3.1.3	Data Flow Diagram of Multi-Fuzzer Platform for Automated Vulnerability Assessments	45

Figure 3.2.1	System Architecture Diagram of Multi-Fuzzer Platform for Automated Vulnerability Assessments	47
Figure 3.2.2	System Flowchart of Multi-Fuzzer Platform for Automated Vulnerability Assessments	49
Figure 3.2.3	Overview of an AFL++ fuzz testing session	53
Figure 3.2.4	Coverage-Guided Fuzzer Architecture	56
Figure 3.2.5	Detailed Fuzzing Architecture in FOT	58
Figure 4.1.1	Overall Block Diagram of Multi-Fuzzer Platform for Automated Vulnerability Assessments	60
Figure 4.1.2	User Interface Block Diagram	61
Figure 4.1.3	Upload & Control Layer Block Diagram	62
Figure 4.1.4	Fuzzing Engines Block Diagram	62
Figure 4.1.5	Result Aggregation and Crash Processing Block Diagram	63
Figure 4.1.6	CVE-like Report Generation Block Diagram	64
Figure 4.1.7	Reporting and Visualization Block Diagram	65
Figure 4.3.1	Emerging Architecture for LLM Applications	69
Figure 4.3.2	Transformer Architecture	70
Figure 4.3.3	End-to-End Transformation Flow	72
Figure 4.4	System Component Data Flow and Interaction Diagram	80
Figure 5.2.2.1	Ubuntu System Update and Package Installation	86
Figure 5.2.2.2	AFL++ Installation	87
Figure 5.2.2.3	Honggfuzz Installation	87
Figure 5.3.1	Internal Configuration and Binary Paths of AFL++	88
Figure 5.3.2.1	Target Corpus Data Directory	89
Figure 5.3.2.2	Test Cases Corpus Data Directory	89
Figure 5.3.2.3	Output Crash Directory	90
Figure 5.3.2.4	Target Application Directory	90
Figure 5.3.2.5	Test Cases Application Directory	90
Figure 5.3.2.6	AFL++ Crash Logs	91
Figure 5.3.2.7	Honggfuzz Crash Logs	92
Figure 5.3.2.8	Combined Crash Report from Both Fuzzers (Section 1)	92

Figure 5.3.2.9	Combined Crash Report from Both Fuzzers (Section 2)	93
Figure 5.3.2.10	Combined Crash Report from Both Fuzzers (Section 3)	94
Figure 5.3.3.1	Parallel Fuzzing Execution Script	95
Figure 5.3.3.2	AFL++ Information Extracting Script	96
Figure 5.3.3.3	Honggfuzz Information Extracting Script	96
Figure 5.3.3.4	Full Crash Information Compiling Script	97
Figure 5.3.3.5	Mitigation Trigger Script	98
Figure 5.3.4.1	File Upload Handling Code	99
Figure 5.3.4.2	JSON Format Conversion Code	100
Figure 5.3.4.3	Request sent to ChatGPT API (OpenAI API) Code	101
Figure 5.3.5	System Resources Configuration	102
Figure 5.4.1.1	Login Process using Username and Password	103
Figure 5.4.1.2	Login Process Unsuccessful	104
Figure 5.4.1.3	User Registration Process	104
Figure 5.4.1.4	User Registration Process with Duplicated Registration	105
Figure 5.4.1.5	Two-factor Authentication Page	105
Figure 5.4.2.1	Multi-Fuzzer Chatbot System Main Interface	106
Figure 5.4.2.2	Binary File Upload Process 1	106
Figure 5.4.2.3	Binary File Upload Process 2	107
Figure 5.4.2.4	Binary File Upload Process 3	107
Figure 5.4.3	Fuzzing Process Begins	108
Figure 5.4.4.1	Crash Detected from Fuzzing Process	109
Figure 5.4.4.2	Low Disk Space Warning	109
Figure 5.4.5.1	CVE Report for testdisk (Section 1) Generated by the LLM	110
Figure 5.4.5.2	CVE Report for testdisk (Section 2) Generated by the LLM	111
Figure 5.4.5.3	CVE Report for testdisk (Section 3) Generated by the LLM	111
Figure 5.4.5.4	CVE Report for testfile (Section 1) Generated by the LLM	112
Figure 5.4.5.5	CVE Report for testfile (Section 2) Generated by the LLM	112

Figure 5.4.5.6	CVE Report for testfile (Section 3) Generated by the LLM	113
Figure 5.4.5.7	CVE Report for testfile (Section 4) Generated by the LLM	113
Figure 5.4.6.1	Confirmation Pop-up for Mitigation Script Installation	114
Figure 5.4.6.2	Mitigation Script Installation Loading Status	114
Figure 5.4.6.3	Mitigation Script Installation Successful	115
Figure 5.4.7.1	Disk Flooding Attack (Before Effect)	116
Figure 5.4.7.2	Disk Flooding Attack (After Effect)	116
Figure 5.4.7.3	Before/After Effect of Disk Flooding Attack	116
Figure 5.4.7.4	File Inode Exhaustion (Before Effect)	117
Figure 5.4.7.5	File Inode Exhaustion (After Effect)	117
Figure 5.4.7.6	Before/After Effect of File Inode Exhaustion	117
Figure 5.4.8.1	Further Question Input Box with Response (Section 1)	118
Figure 5.4.8.2	Further Question Input Box with Response (Section 2)	119
Figure 5.4.9.1	Admin Login Prompt	120
Figure 5.4.9.2	Debugging Dashboard with Command Input Area	120
Figure 5.4.9.3	Force-stop Fuzzing Jobs	120
Figure 5.4.9.4	Debugging Fuzzing Process	121
Figure 5.4.10.1	Account Setting Page with 2FA Enabled	122
Figure 5.4.10.2	Account Setting Page: 2FA Activation Process (Section 1)	122
Figure 5.4.10.3	Account Setting Page: 2FA Activation Process (Section 1)	123
Figure 5.4.10.4	Account Setting Page: 2FA Activated Successfully	123
Figure 5.4.11	Logout Confirmation Prompt	124
Figure 6.2.1	Number of Unique Crash Detection (Target Binaries) using AFL++	143
Figure 6.2.2	Number of Unique Crash Detection (Target Binaries) using Honggfuzz	144
Figure 6.2.3	Conceptual Venn Diagram showing comparative crash-related capabilities between AFL++ and Honggfuzz	150
Figure 6.2.4	CVE-like Response Generated	157



Figure 6.2.5	Recommended Mitigation by the LLM	158
Figure 6.2.6	Chatbot Interaction Testing Bar Chart	159

## LIST OF TABLES

<b>Table Number</b>	<b>Title</b>	<b>Page</b>
Table 2.1.1	Differences between SAST and DAST	10
Table 2.1.2	Differences between NIST and CIS	11
Table 2.1.3	Strengths and Weaknesses of Previous Work & Comparison with the Proposed Solutions	13
Table 2.2.1	Performance Comparison of Testing Techniques	21
Table 2.2.2	Comparison of Generation-Based Fuzzers and Mutation-Based Fuzzers	23
Table 2.2.3	Common White Box, Gray Box and Black Box Fuzzers	26
Table 2.2.4	Pros & Cons of Dumb Fuzzing	28
Table 2.2.5	Pros & Cons of Smart Fuzzing	29
Table 2.2.6	Comparison of different fuzzing techniques	33
Table 5.1	Specifications of Laptop	84
Table 6.1.1	SIGSEGV Vulnerabilities	135
Table 6.1.2	SIGALRM Vulnerabilities	135
Table 6.1.3	SIGABRT Vulnerabilities	136
Table 6.1.4	SIGILL Vulnerabilities	137
Table 6.1.5	Expert Label vs Current Work Evaluation	140
Table 6.2.1	Results Of Fuzzing Tests on Test Cases	145
Table 6.2.2	Execution Time for Crash Detection (Target Binaries) using AFL++	147
Table 6.2.3	Execution Time for Crash Detection (Target Binaries) using Honggfuzz	148
Table 6.2.4	Crash Detection Time of Custom Test Cases by Both Fuzzers	149

## LIST OF ABBREVIATIONS

<i>AFL++</i>	American Fuzzy Lop++
<i>CVE</i>	Common Vulnerabilities and Exposures
<i>DoS</i>	Denial of Service
<i>ELF</i>	Executable and Linkable Format
<i>OSINT</i>	Open-Source Intelligence
<i>PC</i>	Program Counter
<i>SDK</i>	Software Development Kit
<i>SIGSEGV</i>	Signal Segmentation Violation
<i>SIGILL</i>	Signal Illegal Instruction
<i>SQL</i>	Structured Query Language
<i>GDB</i>	GNU Debugger
<i>QEMU</i>	Quick Emulator
<i>CTF</i>	Capture The Flag
<i>RNG</i>	Random Number Generator
<i>UI</i>	User Interface
<i>DLL</i>	Dynamic-Link Library
<i>OS</i>	Operating System
<i>IoT</i>	Internet of Things
<i>LLM</i>	Large Language Model
<i>FOT</i>	Fuzzing Optimization Tool

# CHAPTER 1

## Introduction

Most cybersecurity vulnerabilities stem from unpatched software, poor coding practices, and inadequate vulnerability management processes, which expose systems to potential attacks. In response to these challenges, this project aims to enhance vulnerability assessment techniques by developing a platform that employs multi-fuzzer methodologies for automated vulnerability discovery. This approach integrates various advanced fuzzing tools, such as AFL++ and Honggfuzz, to improve the efficiency and accuracy of vulnerability detection in software applications. The project focuses on automating discovering and analyzing vulnerabilities and delivering a more reliable solution for detecting security flaws. By leveraging automated and diverse fuzzing techniques, this work contributes to advancing cybersecurity practices and offers a scalable, efficient solution for software security testing.

### 1.1 Problem Statement and Motivation

In the current cybersecurity landscape, software applications are increasingly vulnerable to sophisticated attacks that exploit unknown or unpatched security flaws. Traditional methods for vulnerability assessment, which rely heavily on manual testing and a limited set of automated tools, fall short of effectively managing the growing complexity and volume of these threats. Manual testing is not only labour-intensive and time-consuming but also susceptible to human error, which can lead to the oversight of critical vulnerabilities [1]. Additionally, existing automated tools cannot often effectively integrate multiple fuzzing techniques, resulting in incomplete coverage and delayed detection of potential security weaknesses [2]. This poses a significant challenge for organizations that need to maintain strong security postures and protect sensitive information from evolving cyber threats.

The motivation for this project stems from the urgent need to improve the efficiency and effectiveness of vulnerability assessments in software applications. By developing a system based on multi-fuzzer techniques for automated vulnerability assessments, this

project aims to overcome the limitations of current approaches by offering a more comprehensive and integrated solution. The proposed system allows users to upload their software applications via a user-friendly chatbot interface, which then applies various fuzzing tools to thoroughly assess vulnerabilities. This automated approach not only speeds up the vulnerability discovery process but also enhances the accuracy and depth of analysis, ensuring that even the most elusive vulnerabilities are identified and addressed. The project's motivation is anchored in advancing cybersecurity practices by making vulnerability assessments more reliable and scalable. Through automation from software input to the generation of detailed CVEs and exploitation analyses, the project seeks to empower organizations to proactively safeguard against emerging threats and reduce the likelihood of successful cyberattacks.

### 1.2 Objectives

- **To design a multi-fuzzing technique to automate bug and exploitation detection**

The primary objective is to develop a novel multi-fuzzing approach that integrates multiple fuzzing tools, such as AFL++ and Honggfuzz, to automate the detection of software bugs and potential exploitation paths. By combining the unique strengths of these fuzzers, such as AFL++'s coverage-guided fuzzing and Honggfuzz's aggressive mutation strategies, the proposed technique aims to enhance the overall effectiveness of vulnerabilities detection [3]. This approach will automate the entire process, from initiating fuzzing campaigns to analyzing results, thereby reducing the manual effort required in traditional vulnerability assessment methods. The goal is to increase the depth and breadth of vulnerability coverage, uncovering a wider range of potential security flaws that could be exploited by malicious actors.

- **To develop a pipeline to transform bug reports into actionable vulnerabilities similar to CVE using a pretrained LLM**

This objective focuses on creating an automated pipeline that processes and analyzes bug reports generated by the multi-fuzzing technique. Leveraging a pretrained Large

Language Model, the pipeline will be capable of understanding the context and details of the bug reports, automatically categorizing them into actionable vulnerabilities similar to those listed in the Common Vulnerabilities and Exposures (CVE) database. The use of LLMs enables sophisticated natural language processing to interpret complex bug descriptions and map them to known vulnerability patterns. This pipeline aims to bridge the gap between raw fuzzing outputs and meaningful, actionable intelligence for cybersecurity professionals, facilitating quicker and more efficient vulnerability management.

- **To develop a framework to mitigate automatically detected vulnerabilities**

Beyond detection, the project aims to build a comprehensive framework that not only identifies but also mitigates the vulnerabilities detected through the multi-fuzzing process. This framework will integrate with existing security infrastructures and tools to provide automated or semi-automated suggestions for mitigating the discovered vulnerabilities. These suggestions could include patching the software, modifying configurations, or applying other security controls. The framework's objective is to enhance the resilience of software applications by providing a proactive security measure that can adapt to newly detected threats in real-time, reducing the window of opportunity for attackers and enhancing the overall security posture of the organization.

### 1.3 Project Scope and Direction

The scope of this project encompasses the design, development, and deployment of a comprehensive multi-fuzzing platform for automated vulnerability assessment in software applications. The platform aims to integrate two powerful and complementary fuzzing tools: AFL++, known for its coverage-guided fuzzing capabilities, and Honggfuzz, which employs aggressive input mutation and dynamic instrumentation. By combining their unique strengths, the platform seeks to provide broad and deep vulnerability coverage, uncovering both common and complex software flaws.

This system will operate within a controlled testing environment on Ubuntu 24.04, where the fuzzers can run in parallel or in sequence against target binaries submitted by

users. This setup allows for efficient orchestration, resource management, and centralized result aggregation, enabling comparative analysis of fuzzing outputs to evaluate their effectiveness and complementarity.

To enhance usability and utility, the project will also develop a data processing pipeline powered by a pretrained Large Language Model (LLM) [4]. This pipeline will convert raw bug reports generated by fuzzers into actionable vulnerability insights, structured similarly to entries in the Common Vulnerabilities and Exposures (CVE) database. By applying natural language processing to the fuzzers' outputs, the system bridges the gap between raw test results and human-readable intelligence, thus aiding cybersecurity analysts and developers in faster decision-making.

Additionally, the project direction extends toward automated mitigation, closing the loop between detection and remediation. A dedicated framework will be built to analyze the nature of detected vulnerabilities and suggest or perform appropriate mitigation steps—such as patch deployment, configuration adjustments, or enforcement of specific security policies. This will be achieved by integrating the framework with existing software security infrastructures, providing a proactive and adaptable security posture.

Ultimately, this project will move in the direction of creating a scalable, modular, and user-friendly solution that allows users to:

- Upload software via a chatbot interface,
- Initiate a multi-fuzzing assessment,
- Receive structured CVE-like reports, and
- Apply suggested mitigations automatically or manually.

This end-to-end pipeline aims to streamline the vulnerability assessment lifecycle, improve assessment accuracy, and reduce the time and resources typically required in manual processes. The project direction also sets the stage for future extensibility, such as supporting more fuzzers, incorporating AI-assisted exploit generation, or integrating with DevSecOps pipelines.

### 1.4 Contribution

This project makes several meaningful contributions to the field of automated software security testing by designing and developing a multi-fuzzer vulnerability assessment platform that addresses key limitations in current vulnerability discovery practices.

Firstly, the project introduces a multi-fuzzing orchestration framework that integrates AFL++ and Honggfuzz, two state-of-the-art fuzzers with complementary strategies. AFL++ provides coverage-guided fuzzing using a genetic algorithm, while Honggfuzz applies aggressive input mutation and dynamic analysis. By coordinating these fuzzers within a shared environment, the system significantly improves the depth and diversity of vulnerability detection, uncovering edge cases and complex bugs that may be missed when using a single fuzzer alone.

Secondly, the project contributes an innovative automated analysis pipeline that leverages a pretrained Large Language Model (LLM) to convert raw, technical fuzzing outputs into actionable vulnerability descriptions. These descriptions are modeled after the format used by the Common Vulnerabilities and Exposures (CVE) database [5], making them highly usable for developers and security teams. This not only improves the clarity and accessibility of bug information but also supports more efficient vulnerability triage and classification.

Thirdly, the platform incorporates a vulnerability mitigation framework designed to automate the next step after detection, which is remediation. It can suggest or apply countermeasures such as configuration changes, security controls, or patch recommendations. This integrated approach strengthens the defensive lifecycle, reducing the response time between identifying and addressing security issues.

Additionally, the project includes a chatbot-driven user interface that streamlines the interaction between users and the platform. This contribution makes the system more accessible to users with varying levels of technical expertise, enhancing usability and promoting adoption in both educational and professional environments.

Overall, the platform provides a complete and scalable solution that advances current practices in vulnerability assessment by automating fuzzing, reporting, and mitigation.



### 1.5 Report Organization

This report is structured into seven comprehensive chapters to present the development and evaluation of the automated multi-fuzzer vulnerability assessment system.

- **Chapter 1: Introduction** provides an overview of the problem, outlines the project objectives, defines the project scope, and highlights the key contributions of the work.
- **Chapter 2: Literature Review** surveys previous studies and background information on vulnerabilities assessment, fuzzing techniques, and the integration of symbolic and concolic execution in software testing.
- **Chapter 3: System Methodology/Approach** describes the overall design methodology used in the project. It includes architectural diagrams, flowcharts, data flow diagrams, and behavioral models such as use case and activity diagrams, as well as the internal logic of the fuzzing system.
- **Chapter 4: System Design** details the system block diagram, component specifications, LLM architecture, CVE generation design, and the interaction among system modules. This chapter explains how the data flows from the user interface to the final mitigation execution.
- **Chapter 5: System Implementation** outlines the actual hardware and software configurations, environmental setup, and integration procedures. It explains the implementation of each system feature, discusses encountered challenges, and how they were addressed during development.
- **Chapter 6: System Evaluation and Discussion** presents the testing procedures, performance metrics, crash analysis results, evaluation of project objectives, and a comparison between expert evaluation and current automated results. It also discusses key challenges and limitations.
- **Chapter 7: Conclusion and Recommendation** concludes the report by summarizing the outcomes of the project and proposing future improvements and extensions to enhance system capability and usability.

## CHAPTER 2

### Literature Review

The literature review section offers a detailed examination of methods used in vulnerability assessment, beginning with a survey of traditional and contemporary tools designed to detect security vulnerabilities. It introduces fuzzing as a dynamic testing approach, recognized for its capacity to identify concealed flaws in software, and explains its integration with symbolic and concolic execution to strengthen testing processes. Additionally, the review explores into specific fuzzing techniques and their role in identifying various types of vulnerabilities, emphasizing their practical contribution to enhancing the security of software systems.

#### 2.1 Previous Works on Vulnerabilities Assessment

In the evolving landscape of cybersecurity, a range of tools and methodologies have been developed to detect vulnerabilities and secure software applications. This literature review explores previous works on vulnerability detection, focusing on OSINT tools, network vulnerability scanners, web application vulnerability scanners, Static Application Security Testing (SAST) tools, Dynamic Application Security Testing (DAST) tools, configuration management and compliance tools, and container and cloud security tools. This section evaluates the strengths and weaknesses of these tools and compares them with the proposed automated multi-fuzzer platform in table.

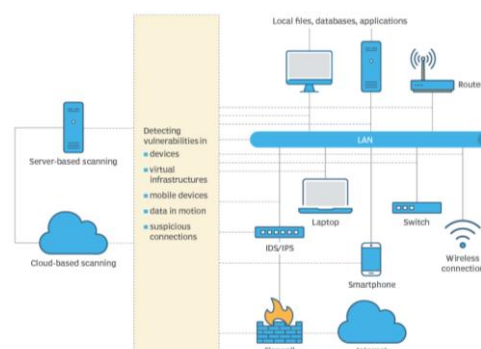
**Open Source Intelligence (OSINT) tools** play a significant role in vulnerability assessment by collecting and analyzing publicly available information from various sources to identify potential security threats [6]. These tools go through the internet, including social media platforms, forums, blogs, and publicly accessible databases, to uncover sensitive information such as exposed credentials, misconfigured services, and publicly known vulnerabilities that outsiders could exploit. They can reveal valuable insights about an organization's digital footprint, including potential points of entry for attackers, and are often used in conjunction with other security tools to provide a more comprehensive assessment of an organization's security posture. [7, Figure 2.1.1]

illustrates the effectiveness of OSINT in the reconnaissance phase of penetration testing, where it facilitates the thorough collection of publicly available information.



**Figure 2.1.1:** Process of Penetration Testing

**Network vulnerability scanners** are designed to identify vulnerabilities within networked systems, such as routers, switches, servers, and other network devices. These tools work by scanning devices for open ports, misconfigurations, missing patches, outdated software versions, and other known vulnerabilities. Network scanners, such as Nessus, OpenVAS, and Nmap, are commonly used in network security assessments to detect weaknesses that could be exploited by attackers to gain unauthorized access or disrupt services [8]. By providing a comprehensive view of an organization's network infrastructure, these tools help security professionals understand the exposure to external threats and take proactive measures to secure network components. They are also valuable in routine security checks, as they help ensure that network devices remain secure and compliant with security policies. [9, Figure 2.1.2] shows how a vulnerability scanner works with different components in detecting vulnerabilities.



**Figure 2.1.2:** How does a vulnerability scanner work?

**Web application vulnerability scanners** are specialized tools that focus on identifying security vulnerabilities within web applications. These scanners, including tools like OWASP ZAP, Burp Suite, and Acunetix, are designed to test web applications for common vulnerabilities such as SQL injection, cross-site scripting (XSS), cross-site request forgery (CSRF), and other security flaws unique to web applications [10]. By simulating attacks and analyzing the application's response, these tools can detect vulnerabilities that could be exploited by attackers to gain unauthorized access, manipulate data, or disrupt web services. Web application scanners are particularly valuable for organizations that rely heavily on web-based services, as they help ensure that these services are secure against the ever-evolving landscape of web-based threats [11]. These scanners are often integrated with continuous integration/continuous deployment (CI/CD) pipelines to provide ongoing security assessments as web applications are developed and updated. By examining code during development and deployment, security vulnerabilities can be detected and resolved quickly, minimizing the risk of exploitation. CI/CD in [12, Figure 2.1.3] involves a set of automated workflows, ranging from code creation to deployment in production, which facilitate the consistent and rapid delivery of code updates to the production environment.



**Figure 2.1.3:** Various steps in the CI/CD pipeline

**Static Application Security Testing (SAST) tools** are used to analyze source code, binaries, or bytecode to identify security vulnerabilities without executing the program [13]. These tools, such as SonarQube, Checkmarx, and Fortify Static Code Analyzer, scan the code for security flaws like buffer overflows, SQL injection, cross-site scripting, and insecure cryptographic practices. SAST tools are typically employed early in the software development lifecycle, allowing developers to detect and address security issues before the software is deployed. By examining the codebase, these tools

can identify potential vulnerabilities that might not be apparent through dynamic analysis. SAST tools are valuable for organizations seeking to integrate security into their development processes (a practice known as "shift-left" security) by catching vulnerabilities early when they are easier and less costly to fix [14].

**Dynamic Application Security Testing (DAST) tools** are designed to identify vulnerabilities in running applications by simulating attacks and monitoring the application's behavior. Unlike SAST tools, which analyze code statically, DAST tools like AppScan, WebInspect, and Netsparker operate dynamically by interacting with a live application [15]. They test for runtime vulnerabilities such as input validation errors, server misconfigurations, authentication issues, and other flaws that only show during execution. DAST tools are particularly useful for identifying security issues that arise from the interaction of different components or from the runtime environment. By conducting tests on a live application, DAST tools can provide a more comprehensive view of an application's security posture, complementing the findings of SAST tools and offering insights into vulnerabilities that are only detectable in a live environment. [16, Table 2.1.1] below are the differences between SAST and DAST from different perspectives.

**Table 2.1.1:** Differences between SAST and DAST

Feature	SAST	DAST
<b>Aim</b>	Exposes application weaknesses early in the development cycle.	Exposes weaknesses towards the end of the development cycle.
<b>Requirements</b>	Only the source code	Only the executed software
<b>Technique</b>	White-box without access to the framework or internal design	Black-box, with access to the framework and internal design.
<b>Compatibility</b>	All applications	Only web applications
<b>Approach</b>	Inside-out	Outside-in
<b>Advantage</b>	Less expensive to close weak points	Can be run on different application environments.

**Configuration management and compliance tools** are essential for maintaining secure and compliant IT environments. Tools such as Chef, Puppet, and Ansible are designed to automate the management of system configurations, ensuring that systems are configured according to predefined security policies and compliance standards [17]. These tools help prevent misconfigurations that could lead to security breaches by automating the enforcement of configuration baselines and detecting deviations from these baselines. Additionally, configuration management tools can integrate with vulnerability management platforms to provide a holistic view of an organization's security posture.

They are crucial for organizations that need to maintain continuous compliance with security standards such as the Center for Internet Security (CIS) Benchmarks, the National Institute of Standards and Technology (NIST) guidelines, and other regulatory requirements. NIST standards assist in improving cybersecurity and information security by offering a framework of guidelines to manage risk management processes effectively. CIS provides a collection of best practices designed to help organizations protect themselves against cyber threats. It includes CIS Controls, a prioritized set of measures aimed at safeguarding against common cyberattacks [18]. [18, Table 2.1.2] below shows a comparison of the differences between NIST and CIS from different perspectives.

**Table 2.1.2:** Differences between NIST and CIS

Aspect	NIST	CIS
Approach	Risk-based approach	Actionable, prioritized controls
Focus	Comprehensive cybersecurity framework	Specific, practical security controls
Structure	5 functions: Identify, Protect, Detect, Respond, Recover	20 prioritized controls
Flexibility	Adaptable, suitable for various sectors	Emphasizes quick implementation
Implementation speed	May require more time for full adoption	Quick implementation of actionable controls
Industry usage	Government and public sectors	Small, medium and large enterprises
Updates	Periodic updates and revisions	Community-driven regular updates

As organizations increasingly adopt containerized and cloud-based environments, specialized security tools have been developed to address the unique challenges of these platforms. **Container security tools**, such as Aqua Security Platform, Sysdig Secure, and Aikido Container Security are designed to monitor container activity, enforce security policies, and detect anomalies in containerized applications [19]. These tools provide visibility into container runtime behavior, helping organizations detect and respond to security incidents in real-time. **Cloud security tools**, such as AWS Inspector and Azure Security Center, offer comprehensive assessments of cloud environments, helping organizations secure their cloud infrastructure against threats such as misconfigurations, insecure interfaces, and unauthorized access [20]. These tools are particularly important in dynamic cloud environments where infrastructure changes frequently and security policies must adapt quickly to ensure the protection of sensitive data and services.

To effectively compare various vulnerability assessment techniques and their relevance to our proposed multi-fuzzer solution, the following table summarizes the **strengths and weaknesses** of existing methods. Each technique has its advantages and limitations, which are critically analyzed to highlight how our solution builds upon and improves these approaches. This comparison highlights the added value of integrating multiple fuzzing techniques into a unified automated platform, providing a more comprehensive and efficient solution for vulnerability detection.

**Table 2.1.3:** Strengths and Weaknesses of Previous Work & Comparison with the Proposed Solutions

<b>Technique</b>	<b>Strengths</b>	<b>Weaknesses</b>	<b>Comparison with the Proposed Solution</b>
<b>OSINT Tools</b>	Effective in identifying exposed credentials and misconfigurations from public data.	Requires manual filtering and limited integration with automated tools.	The proposed solution automates OSINT data integration, improving efficiency and relevance in vulnerability detection.
<b>Network Vulnerability Scanners</b>	Comprehensive network security assessment and high effective for known vulnerabilities.	High false positives and limited in detecting zero-days.	Multi-fuzzer techniques reduce false positives and enhance zero-day detection capabilities.
<b>Web Application Scanners</b>	Detects web-specific vulnerabilities effectively with integration of CI/CD pipelines.	Limited in handling complex logic vulnerabilities and it depends on frequent updates.	The proposed solution addresses complex logic and business-layer vulnerabilities more effectively through advanced fuzzing.
<b>SAST Tools</b>	Early detection of vulnerabilities and integration of security in development (shift-left).	Misses runtime vulnerabilities with high false positives from static analysis.	The multi-fuzzer platform combines static and dynamic analysis, covering both compile-time and runtime vulnerabilities, reducing false positives.
<b>DAST Tools</b>	Identifies runtime vulnerabilities through simulated attacks on live applications.	Resource-intensive and limited to vulnerabilities visible during testing sessions.	The proposed solution offers comprehensive vulnerability detection across various runtime conditions beyond standard DAST capabilities.



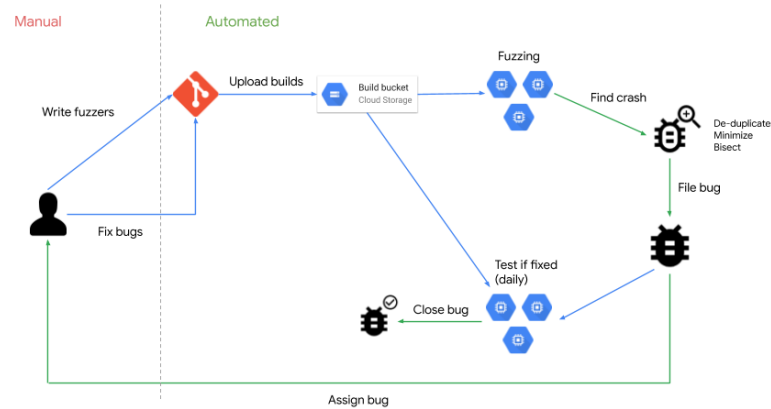
<b>Configuration Management Tools</b>	Ensures secure system configurations and continuous compliance.	Focuses on configuration, not direct vulnerability detection and limited integration with other tools.	The multi-fuzzer platform integrates configuration checks with broader vulnerability assessments, offering a more comprehensive security solution.
<b>Container and Cloud Security Tools</b>	Specialized in securing dynamic, scalable environments like containers and cloud platforms.	Challenges with integrating traditional tools and complex security management across hybrid environments.	The proposed solution provides unified vulnerability assessment across diverse environments using multi-fuzzing techniques tailored for different deployment scenarios.

### 2.2 Fuzzing

#### 2.2.1 Introduction to Fuzzing

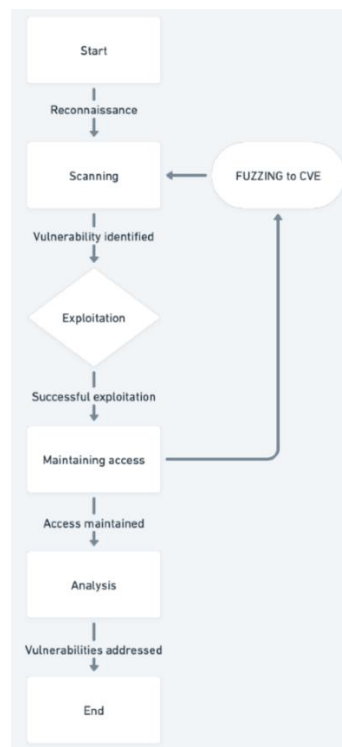
Fuzzing, also known as fuzz testing, is an active testing method that involves providing arbitrary data to a program repeatedly until it encounters a failure or crash. Barton Miller at the University of Wisconsin pioneered this technique in the late 1980s [21]. Since its inception, fuzz testing has demonstrated its effectiveness in identifying software vulnerabilities. Initially rooted in randomly generated test data, also known as random fuzzing, the evolution of symbolic computation, model-based testing, and dynamic test case generation has paved the way for more sophisticated fuzzing methodologies. Takanen et al. [22] extensively cover fuzzing techniques in their book, offering detailed insights with statistical data and comprehensive case studies. Fuzzers are categorized as either generation-based or mutation-based, depending on whether test inputs are created from scratch or via modification of existing ones.

Additionally, these tools are categorized as whitebox, greybox, or black-box fuzzers. Whitebox fuzzers specialize in tracing execution paths and managing complex constraints through intensive program analysis [23]. In contrast, greybox fuzzing tools employ lighter program analysis methods to improve code coverage [24]. As of February 2023, ClusterFuzz has detected approximately 27,000 bugs within Google applications such as Chrome. Moreover, it has contributed to the identification and resolution of more than 8,900 vulnerabilities and 28,000 bugs across 850 projects integrated with OSS-Fuzz [25]. [25, Figure 2.2.1] below shows its overview of scalable fuzzing infrastructure that finds security and stability issues in software. It offers numerous functionalities that facilitate the smooth integration of fuzzing into the development process of a software project. Conversely, black-box fuzzers solely observe input/output behavior during execution and do not require access to the source code of the Program Under Test (PUT) [26].



**Figure 2.2.1:** Overview of Scalable Fuzzing Infrastructure

Figure 2.2.2 below shows the fuzzing process initiated with reconnaissance, scanning the system to identify vulnerabilities. Once found, exploitation techniques are employed to capitalize on these weaknesses. Successful exploitation grants ongoing access to the system, enabling deeper analysis and exploration. Fuzzing techniques, designed to detect Common Vulnerabilities and Exposures (CVEs), are utilized during the scanning phase to pinpoint potential security flaws within the system.



**Figure 2.2.2:** Overall Process of Fuzzing

### 2.2.2 The Synergy of Fuzzing, Symbolic, and Concolic Execution in Software Testing

In the ever-evolving landscape of software development and cybersecurity, the quest to fortify systems against vulnerabilities and bugs remains a paramount challenge. Fuzzing, symbolic execution, and concolic execution emerge as pivotal techniques, each offering distinct yet interconnected approaches to identify flaws and enhance software resilience. Fuzzing, the venerable technique born from Barton Miller's pioneering work, stands as a cornerstone in software testing. It employs the injection of random or unexpected data into a system to probe for vulnerabilities, crashes, or unpredictable behavior. However, as systems grew in complexity, the need for more sophisticated methods arose.

Symbolic execution stepped onto the stage, revolutionizing software analysis by exploring program paths using symbolic values instead of concrete inputs. It navigates through various execution paths, unveiling potential vulnerabilities and constraints without executing the actual program. Symbolic execution's skill lies in its ability to analyze multiple paths simultaneously, providing valuable insights into program behaviors and identifying hard-to-reach bugs. [27, Figure 2.2.3] shows the overall process of generating test inputs in Symbolic Execution (SE). This method begins by taking the Program Under Test (PUT) and the initial test-input (ITI) as inputs and yields new test inputs by resolving constraints. Initially, SE loads the PUT and establishes its Control Flow Graph (CFG), extracting all basic block addresses. Subsequently, the PUT is dynamically executed with the initial test-input (ITI), determining the covered basic block addresses by correlating the path to the CFG. If the basic block (BB) in the set of AllBBAddr does not correspond with the BB in CoveredBBAddr, it is then included in CONS for further processing. Eventually, leveraging the Z3 solver, SE generates a new effective test-input (NTI) by resolving the constraints accumulated in NTI during this process.

```

Input:   $\mathbb{P}UT$  (Program Under Test)

Input:   $\mathbb{I}_{TI}$  (Initial Test-Input)

1:   $T_P \leftarrow \text{Load}(\mathbb{P}UT)$ 

2:   $CFG \leftarrow \text{cfgConstruction}(\mathbb{P}UT)$ 

3:   $AllBBAddr \leftarrow \text{ExtractAllBBAddr}(CFG)$ 

4:   $Path \leftarrow \text{DynExecution}(\mathbb{P}UT, \mathbb{I}_{TI})$ 

5:   $CoveredBBAddr \leftarrow \text{PathMapping}(Path, CFG)$ 

6:  foreach  $BB \in AllBBAddr$  do

7:    if  $BB \notin CoveredBBAddr$  do

8:       $CONS \leftarrow \text{ConstraintCollection}(BB)$ 

9:       $\mathbb{N}_{TI} \leftarrow \text{Solver}(CONS)$ 

10:    else

11:      Continue

12:    end if

13:  end for

Output:  $\mathbb{N}_{TI}$  (New Test-Input)

```

**Figure 2.2.3:** Test-Input Generation of Symbolic Execution Algorithm

Concolic execution bridges the gap between symbolic and concrete execution. This technique combines actual inputs with symbolic representations, exploring different program paths while leveraging the strengths of both approaches. By using concrete inputs to execute a program and concurrently tracking the symbolic representation of these inputs, concolic execution efficiently explores multiple execution paths, enhancing bug detection and program verification. [28, Figure 2.2.4] below shows how the `test_h()` function explores different execution paths within function `h(int x, int y)` using concolic execution. It iterates through different inputs based on the path constraints until all possible paths have been covered or constraints have been satisfied. The `execute_h()` function simulates the execution of `h(int x, int y)` with given inputs and records the path constraints. The program stops when all predicates have been negated or when an error condition is met within the function `h()`.

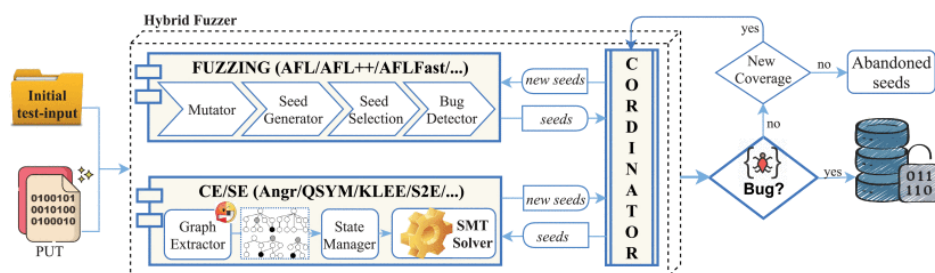
```

00 int f(int x) {return 2 * x;}
01 int h(int x, int y) {
02     if (x != y)
03         if (f(x) == x + 10)
04             error();
05     return 0;
06 }

```

**Figure 2.2.4:** Concolic Execution Exploration in Function h(int x, int y)

Hybrid fuzzing emerges as the convergence point of these techniques. It merges the diversity of fuzzing with the intelligent path exploration of symbolic or concolic execution. This hybrid approach leverages fuzzing's ability to generate varied inputs with the guidance of symbolic or concolic techniques. By intelligently steering the generation of inputs towards specific code paths or conditions, hybrid fuzzing optimizes bug detection, enhancing the effectiveness of software testing. [27, Figure 2.2.5] below illustrates the comprehensive workflow of hybrid fuzzing, comprising three primary elements: the fuzzer, CE engine, and a coordinator. Acting as an intermediary, the coordinator orchestrates the functioning of fuzzing and CE techniques, undertaking three primary responsibilities. Firstly, it monitors the fuzzer, deciding when to initiate the CE engine. Secondly, it sets up the operational environment for CE. Thirdly, it selects and filters test inputs for execution between the fuzzer and CE. Initially, the coordinator's test-input selection module identifies the test-input file in the fuzzer's queue to be transmitted first to the CE engine. Before commencing CE, the coordinator organizes the test inputs in the fuzzer's queue based on their effectiveness.



**Figure 2.2.5:** Hybrid Fuzzing Technique's High-Level Architecture

The interconnectedness of these techniques marks a paradigm shift in software testing methodologies. Fuzzing, symbolic, and concolic execution complement each other, offering a holistic approach to software analysis. They collectively contribute to uncovering vulnerabilities, enhancing code coverage, and fortifying systems against potential threats.

[29, Figure 2.2.6] involves employing the Critical Branch Selection (CBS) and Priority Score Calculation (PSC) algorithms to transform the initial input list into an ordered input list. The algorithm evaluates branch criticality using three metrics: hit (H) to determine if a branch is explored, solvability (S) to ascertain if the branch can be solved, and complexity (C) indicating the difficulty of solving the branch. Further elaboration will be provided on these metrics. Additionally, the algorithm generates a priority score for each input based on the target branches. These scores guide the selection of inputs for the ordered input list. The selection process involves considering how critical a branch is and how likely it can be solved, ultimately producing an ordered input list. Detailed explanations on branch criticality assessment and the determination of priority scores will follow in this section.

```

Input: input_list
Output: ordering_input_list

1:  for each input in input_list do
2:    branches = get_branches(input)
3:    for each branch in branches do
4:      H = if_already_hit(branch)
5:      S = get_solvability(branch)
6:      if H == 0 and S == 1 then
7:        target_branch_list.append(branch)
8:      end if
9:    end for
10:   for branch in target_branch_list do
11:     C[branch] = get_complexity(branch)
12:   end for
13:   sort C by inverted order
14:   final_branch_list = {select top 50% branches from C with most complexity}
15:   input_score = get_input_score(final_branch_list)
16:   priority_score_dict[input] = input_score
17: end for
18: ordering_input_list = sort priority_score_dict by score
19: return ordering_input_list

```

**Figure 2.2.6:** Selective Hybrid Fuzzing Approach (CBS and PSC algorithms)

[27, Table 2.2.1] below presents a performance analysis of various software testing methodologies, indicating that hybrid fuzzing techniques outperform other methods. Moreover, the prevalence of hybrid fuzzing tools in numerous software testing competitions highlights their significant effectiveness in current testing practices.

**Table 2.2.1:** Performance Comparison of Testing Techniques

Testing Approaches	Usage	Accuracy	Bug detection	Scalability
Static	Easy	Low	Moderate	Good
Dynamic	Hard	High	Easy	Good
SE/CE	Hard	High	Moderate	Bad
Fuzzing	Easy	Moderate	Easy	Good
Hybrid Fuzzing	Easy	Highest	Easy	Best



### 2.2.3 Fuzzing Techniques for Vulnerabilities Detection

Mutation fuzzing involves altering existing valid inputs to generate new test cases. Techniques like bit flipping, byte flipping, segment deletion, splicing, and segment repetition are applied to modify inputs [30]. This approach does not require specific knowledge about the target program and is often termed 'dumb fuzzing.' Despite its simplicity, it can effectively reveal vulnerabilities such as buffer overflows, where an input overrun allocated memory, leading to potential system crashes or execution of malicious code. Additionally, it can uncover format string vulnerabilities where input formatting functions can be manipulated to access unauthorized data or execute arbitrary code. [31, Figure 2.2.7] shows a random mutation scheme, a technique commonly utilized in existing fuzzers like those belonging to the AFL family. This scheme alters a provided seed in a randomized manner.

```

Input: seed – a test case to be mutated
Output: input – a new input to be tested

function RANDOMMUTATION(seed)
  input ← COPYBYTESFROMSEED(seed)
  batch_size ← DECIDEBATCHSIZE()
  for i ← 1 to batch_size do
    mutation ← SELECTOPERATOR()
    pos ← SELECTPOSITION(input)
    input ← APPLYOPERATOR(mutation, input, pos)
  end for
  return input
end function

```

**Figure 2.2.7:** Conventional Random Mutation Scheme Algorithm

Generation fuzzing operates by creating test cases based on defined models or specifications of valid inputs. These models, such as language grammars or binary format specifications, guide the creation of inputs without relying on existing sample inputs. This 'smart fuzzing' technique can expose vulnerabilities like protocol violations, semantic errors, or complex logic flaws. For instance, it can reveal protocol weaknesses or errors in handling specific data structures, leading to security vulnerabilities. The differences between generation-based fuzzers and mutation-based fuzzers are compared in [32, Table 2.2.2] below.

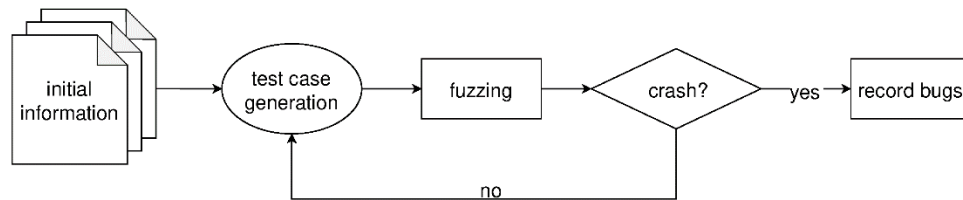
**Table 2.2.2:** Comparison of Generation-Based Fuzzers and Mutation Based-Fuzzers

	Easy to start ?	Priori knowledge	Coverage	Ability to pass validation
Generation based	hard	needed, hard to acquire	high	strong
Mutation based	easy	not needed	low, affected by initial inputs	weak

File format fuzzing focuses on testing applications that handle various file types as input [33]. This technique involves generating modified or malformed versions of files compatible with the targeted application. For example, if testing an image viewer, the fuzzer would create iterations of slightly altered image files, such as JPEGs or PNGs. These files are then sequentially processed by the application, and the fuzzer monitors the software for any crashes, unexpected behaviors, or vulnerabilities triggered by the altered files. This approach is commonly used to assess the resilience of applications like image viewers, document readers, multimedia players, and other software that interacts with specific file types. File format fuzzing aims to uncover weaknesses in how applications handle different file formats, ensuring that they do not crash or expose security vulnerabilities when processing varied inputs.

Protocol fuzzing is focused on testing applications that communicate over networks using various protocols, such as HTTP, FTP, SMTP, or custom network protocols. In this method, the fuzzer generates malformed or altered packets that deviate from the expected structure of these communication protocols. These packets are then sent to the target application, simulating the behavior of a potentially malicious attacker sending corrupted or unexpected data over the network. Protocol fuzzing is effective in uncovering vulnerabilities in network-based software, such as web servers, email servers, FTP clients, or any software that interacts with network protocols. By injecting malformed packets into the communication stream, this technique aims to identify potential security flaws or crashes that could be exploited by attackers sending

malformed or malicious data through network channels. The workflow of network protocol fuzzing is illustrated in [34, Figure 2.2.8], where the initial information is also referred to as the seed.



**Figure 2.2.8:** General Workflow of Network Protocol Fuzzing Techniques

White-box fuzzer operates with complete visibility into the inner workings of the software under test. It accesses the source code, internal APIs, data structures, and other intricate details of the system. With this comprehensive understanding, white-box fuzzers create test inputs that specifically target known paths, functions, or components within the software. By leveraging this deep knowledge, they aim to achieve maximum coverage and thoroughly test various functionalities and code paths. Due to their comprehensive understanding of the software's internal structure, white-box fuzzers are adept at targeting specific paths or functions within the system. They are highly effective at detecting complex vulnerabilities like logic flaws, boundary condition errors, and deep-seated security issues within the software.

[35, Figure 2.2.9] outlines the process of directed model-based whitebox fuzzing, taking inputs such as a program  $P$ , an input model  $M$ , a set of target locations  $L$  within  $P$ , and seed inputs  $T$ . Its primary goal is to generate valid, potentially crashing files that target locations within the program. If no specific targets are supplied, MoWF employs static analysis to identify potential risky areas within the program, such as places prone to null pointer dereferences or divisions by zero (lines 1-3). The algorithm utilizes provided test cases  $T$  as initial inputs for generating tests. In case no seed file is available, MoWF uses the input model  $M$  to create a seed file (lines 4-7).

```

Input: Program  $\mathcal{P}$ , Input Model  $\mathcal{M}$ 
Input: Initial Test Suite  $T$ , Targets  $L$ 
Output: Augmented Test Suite  $T'$ 
1: if  $L = \emptyset$  then
2:    $L \leftarrow \text{IDENTIFYCRITICALLOCATIONS}(\mathcal{P})$ 
3: end if
4: if  $T = \emptyset$  then
5:    $t \leftarrow \text{INstantiateASVALIDINPUT}(\mathcal{M})$ 
6:    $T \leftarrow \{t\}$ 
7: end if
8: while timeout not exceeded do
9:   Target location  $l \leftarrow \text{CHOOSETARGET}(L)$ 
10:  Input file  $t \leftarrow \text{CHOOSEBEST}(T, l)$ 
11:  Fragment Pool  $\Phi \leftarrow \text{FILECRACKER}(T, \mathcal{M})$ 
12:  Crucial IFS  $\Lambda \leftarrow \text{DETECTCRUCIALIFS}(t, l, \mathcal{P}, \mathcal{M})$ 
13:  for all  $\lambda \in \Lambda$  do
14:    Valid files  $T_\lambda \leftarrow \text{FILESTITCHER}(t, \lambda, \Phi, \mathcal{M})$ 
15:    for all  $t_\lambda \in T_\lambda$  that negate  $\lambda$  do
16:      Hybrid file  $\hat{t}_\lambda \leftarrow \text{MARKSYMBOLICVARS}(t_\lambda, \mathcal{M})$ 
17:      Files  $F \leftarrow \text{PATHEXPLORATION}(\hat{t}_\lambda, \lambda, l, L, \mathcal{P})$ 
18:      for all  $f \in F$  do
19:        Valid file  $f' \leftarrow \text{FILEREPAIR}(f, \mathcal{M})$ 
20:         $T \leftarrow T \cup f'$ 
21:      end for
22:    end for
23:  end for
24: end while
25:  $T' \leftarrow T$ 

```

**Figure 2.2.9:** Model-based Whitebox Fuzzing Algorithm

Gray-box fuzzers have a limited but somewhat detailed view of the system. While they lack full access to the source code or internal structures, they possess certain insights into the system's behavior. This partial knowledge might include information about function calls, basic control flows, or limited structural aspects. Using these insights, gray-box fuzzers guide the fuzzing process towards specific areas of interest, attempting to explore potential vulnerabilities within the constraints of their limited visibility. With a limited but somewhat detailed insight into the system's behavior, gray-box fuzzers excel in exploring vulnerabilities related to semi-restricted areas. They are capable of uncovering issues like improper input validation, access control vulnerabilities, and basic memory-related problems. [36, Figure 2.2.10] below shows the coverage-based greybox fuzzing algorithm.

```

Input: Seed Corpus  $S$ 
1: repeat
2:    $s = \text{CHOOSENEXT}(S)$  // Search Strategy
3:    $p = \text{ASSIGNENERGY}(s)$  // Power Schedule
4:   for  $i$  from 1 to  $p$  do
5:      $s' = \text{MUTATE\_INPUT}(s)$ 
6:     if  $s'$  crashes then
7:       add  $s'$  to  $S_x$ 
8:     else if  $\text{ISINTERESTING}(s')$  then
9:       add  $s'$  to  $S$ 
10:    end if
11:  end for
12: until timeout reached or abort-signal
Output: Crashing Inputs  $S_x$ 

```

**Figure 2.2.10:** Coverage-based Greybox Fuzzing Algorithm

Black-box fuzzer operates without any internal knowledge or access to the software's structure or source code. It treats the system as a "black box," interacting with it solely based on input-output behavior. Black-box fuzzers generate a wide range of inputs without understanding the internal mechanisms of the software. Their primary objective is to discover vulnerabilities purely through observed behaviors and unexpected responses from the system. Operating solely based on observed behaviors, black-box fuzzers focus on identifying vulnerabilities through external interactions. They are efficient at discovering common security issues such as input validation errors, buffer overflows, and other surface-level vulnerabilities that can be triggered through unexpected inputs. Some common white box, gray box and black box fuzzers are listed in [32, Table 2.2.3] below. Although blackbox fuzzing can be remarkably effective, its limitations are well known. For example, the “then” branch of the conditional statement in [37, Figure 2.2.11] has only 1 in  $2^{32}$  chances of being exercised if the input variable  $x$  has a randomly chosen 32-bit value. This intuitively explains why blackbox fuzzing usually provides low code coverage and can miss security bugs.

**Table 2.2.3:** Common White Box, Gray Box and Black Box Fuzzers

	White box fuzzers	Gray box fuzzers	Black box fuzzers
Generation based	SPIKE (Bowne 2015), Sulley (Amini 2017), Peach (PeachTech 2017)		
Mutation based	Miller (Takanen et al. 2008)	AFL (Zalewski 2017a), Driller (Stephens et al. 2016), Vuzzer (Rawat et al. 2017), TaintScope (Wang et al. 2010), Mayhem (Cha et al. 2012)	SAGE (Godefroid et al. 2012), Libfuzzer (libfuzzer 2017)

```
int foo(int x) { // x is an input
    int y = x + 3;
    if (y == 13) abort(); // error
    return 0;
}
```

**Figure 2.2.11:** “then” Branch of Conditional Statement

On the other hand, DOM (Document Object Model) fuzzing is a specialized technique utilized specifically in web browser testing, focusing on manipulating the structure and elements of web pages through the browser's DOM API [38]. This method involves using JavaScript to interact with the web page's elements, such as HTML, CSS, and JavaScript code, to create variations and test the browser's behavior. Instead of altering files, DOM fuzzing manipulates the elements within a live web page, performing actions like changing attributes, deleting nodes, rearranging the DOM tree, or triggering garbage collection. The goal is to uncover vulnerabilities related to memory corruption or logic flaws within the browser's rendering engine. While DOM fuzzing allows for testing the behavior of web browsers in response to manipulated web page elements, it can face challenges in reproducing crashes efficiently and identifying the root cause of discovered vulnerabilities, as compared to traditional file-based fuzzing methods. This method is specifically tailored to scrutinize the browser's behavior with dynamic content and user interactions, focusing on the internal handling of web page structures.

[39, Figure 2.2.12] shows the algorithm of merging two DOM trees in freedom.

---

**Algorithm 1:** Merging two DOM trees in FREEDOM.

---

```

Input: Two DOM trees  $T_a$  and  $T_b$  in two documents, an object map
Result:  $T_a$  being enlarged by merging with the nodes in  $T_b$ 
// ObjectMap: a global object map used throughout merging.
1 Procedure mergeElement( $n_a, n_b$ , ObjectMap)
2   TargetSet  $\leftarrow \emptyset$ ;
3   for each  $n \in \text{getOffsprings}(n_a)$  do
4     if  $\text{getType}(n) = \text{getType}(n_b)$  then
5       TargetSet  $\leftarrow \text{TargetSet} \cup \{n\}$ ;
6     end
7   end
8   if TargetSet =  $\emptyset$  then
9     // Move the sub-tree rooted at  $n_b$  to be a child of  $n_a$ .
10    insertChild( $n_a, n_b$ );
11  else
12     $n_t \sim \text{TargetSet}$ ; // Randomly sample a node from the set.
13    mergeAttributesAndText( $n_t, n_b$ );
14    ObjectMap[ $n_b$ ]  $\leftarrow n_t$ ;
15    for each  $n \in \text{getChildren}(n_b)$  do
16      mergeElement( $n_t, n$ , ObjectMap);
17    end
18  end
19 Procedure mergeTree( $T_a, T_b$ , ObjectMap)
20    $r_a \leftarrow \text{getRoot}(T_a)$ ;  $r_b \leftarrow \text{getRoot}(T_b)$ ; // The tree root is the <body> element.
21   for each  $n_b \in \text{getChildren}(r_b)$  do
22     mergeElement( $r_a, n_b$ , ObjectMap);
23   end
24   for each  $n_b \in T_b - \{r_b\}$  do
25     if  $\neg \exists \text{ObjectMap}[n_b]$  then
26       addElementIntoGlobalContext( $n_b$ );
27     end
28   end

```

---

**Figure 2.2.12:** Algorithm: Merging two DOM trees in FREEDOM

API fuzzing involves testing the Application Programming Interfaces (APIs) of software systems by sending unexpected, invalid, or edge-case inputs to the API endpoints [40]. This technique aims to uncover vulnerabilities within the API's functionality, security, and data handling processes. Common vulnerabilities tested through API fuzzing include input validation issues, where unexpected inputs may trigger buffer overflows, injection attacks (like SQL injection or command injection), and authentication or authorization flaws that could lead to unauthorized access or exposure of sensitive data. Additionally, API fuzzing might expose issues related to rate limiting, error handling, or unexpected behavior when handling malformed requests, providing a comprehensive assessment of the API's robustness and security posture.

Dumb Fuzzing operates on a blind and random basis, where inputs are generated without a comprehensive understanding of the target application's structure or expected data formats [41]. The approach involves bombarding the system with a plethora of random, nonsensical, or malformed inputs in the hopes of triggering unexpected behavior. Dumb fuzzing casts a wide net, which can sometimes reveal surface-level vulnerabilities such as crashes, memory leaks, or basic input validation errors. For instance, it might uncover issues like buffer overflows or simple parsing errors that occur due to the system's inability to handle unexpected input formats. However, due to its indiscriminate nature, dumb fuzzing might miss more complex or subtle bugs that require a more tailored approach. Its reliance on randomness means it may not effectively trigger deeper, more intricate flaws within a system's logic or processing. The pros and cons of dumb fuzzing are illustrated in the [41, Table 2.2.4] below.

**Table 2.2.4:** Pros & Cons of Dumb Fuzzing

<b>Dumb fuzzing pros</b>	<b>Dumb fuzzing cons</b>
Straightforward to set up, run, and maintain	Limited code coverage due to the fully randomized input
Requires minimum amount of work for the initial setup	Sometimes, it tests a parser than your program

Smart Fuzzing takes a more targeted and systematic stance toward fuzz testing. Smart fuzzing involves a thorough analysis of the target system's structure, expected inputs, data handling, and potential weak points [41]. By leveraging knowledge about the application's behavior, it crafts inputs that are more likely to uncover specific vulnerabilities. Smart fuzzing is equipped to unveil sophisticated issues that might escape random or blind testing. It can reveal complex parsing errors, protocol-level vulnerabilities, injection attacks (such as SQL injection or command injection), and deeper logic flaws within the software. Furthermore, it is adept at detecting authentication bypasses, access control issues, or scenarios where the system fails under specific, unexpected conditions. Smart fuzzing's ability to generate purposeful, meaningful test cases based on an understanding of the system's behavior makes it more effective at finding intricate vulnerabilities compared to its random counterpart. The pros and cons of dumb fuzzing are illustrated in the [41, Table 2.2.5] below.

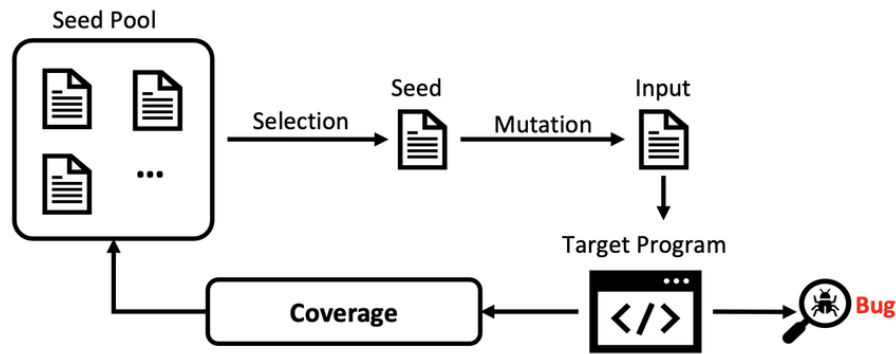
**Table 2.2.5: Pros & Cons of Smart Fuzzing**

Smart fuzzing pros	Smart fuzzing cons
Greater code coverage in comparison with dumb fuzzers	Requires more work to set up, run and maintain
Catches more bugs thanks to greater code coverage	-

On the other hand, hybrid fuzzing, a fusion of multiple fuzzing techniques, combines the strengths of various approaches to bolster overall effectiveness [27]. By merging the exhaustive test case generation of generation fuzzing with the broad randomness and coverage of mutation fuzzing, hybrid fuzzing achieves comprehensive code coverage, effectively targets specific vulnerability classes (such as memory-related issues, protocol violations, and logic errors), and efficiently explores both known and unknown areas of software systems. This multifaceted approach significantly enhances the chances of identifying a diverse array of vulnerabilities and fortifying software against potential security weaknesses.



Coverage-guided fuzzing is a methodology that strategically targets less-exercised portions of a codebase, aiming to unveil vulnerabilities lurking within these under-tested areas. By monitoring and recording the code segments traversed by the fuzzing inputs, this technique prioritizes exploring sections of the software that have had limited exposure. This focused approach is particularly adept at uncovering vulnerabilities that might remain concealed under more conventional testing methodologies. Vulnerabilities such as intricate logic flaws, obscure memory corruptions, or edge-case scenarios within the code structure can often be discovered through coverage-guided fuzzing. Its emphasis on probing less-visited parts of the codebase amplifies the likelihood of exposing vulnerabilities that might evade detection through standard testing practices. The overview of coverage-guided fuzzing is shown in the [42, Figure 2.2.13] below. The following [43, Figure 2.2.14] is the algorithm for coverage-guided fuzzing.



**Figure 2.2.13:** Coverage-guided Fuzzing Overview.

```

1: procedure FUZZ(program  $p$ , set of seed inputs  $I_0$ )
2:    $Inputs \leftarrow I_0$ 
3:    $TotalCoverage \leftarrow$  coverage of  $p$  on  $Inputs$ 
4:   while within time budget do
5:      $i \leftarrow$  pick from  $Inputs$ 
6:      $i' \leftarrow$  mutate  $i$ 
7:      $coverage, error \leftarrow$  execute  $p$  on  $i'$ 
8:     if  $\exists$  error then
9:       report error and faulty input  $i'$ 
10:      optionally, exit
11:    else if  $coverage \not\subseteq TotalCoverage$  then
12:      add  $i'$  to  $Inputs$ 
13:       $TotalCoverage \leftarrow TotalCoverage \cup coverage$ 
  
```

**Figure 2.2.14:** Algorithm for Coverage-Guided Fuzzing

Grammar-based fuzzing is a method that employs predefined grammars to produce inputs adhering to syntactic validity. This approach proves particularly beneficial in identifying bugs stemming from invalid grammatical structures, notably exposing vulnerabilities like SQL injections. By crafting inputs based on grammatical rules, this technique effectively scrutinizes protocols and file formats, often revealing vulnerabilities inherent in these systems. For instance, it can uncover issues arising from malformed structures within protocols or misinterpretations in file formats, highlighting potential entry points for attackers aiming to exploit these weaknesses. [44, Figure 2.2.15] below is the algorithm for fuzzing-based grammar inference.

```

Input : Seed inputs  $I \subseteq \text{validInputs}(p)$ , set of terminals  $\Sigma$  and program  $p$ 
Output: Inferred grammar  $G'_p$ 
1  $G_s := \text{findSeedGrammar}(I, p)$ ;
2  $N_s := G_s.\text{getNonTerminals}()$ ;
3  $G'_p := G_s.\text{clone}()$ ;
4 for  $A \in N_s$  do
5    $c := \text{null}$ ;
6   repeat
7      $M := \text{runNL}^*(p, A, \Sigma, N_s, c)$ ;
8      $c := \text{searchForCounterexample}(M, p, A, G_s, 1000, 10)$   $\triangleright$  See Algorithm 2;
9   until  $c = \text{null}$ ;
10   $\alpha := M.\text{toRegularExpression}()$ ;
11   $G'_p.\text{add}("A \rightarrow \alpha")$ 
12 return  $G'_p$ 

```

**Figure 2.2.15:** Algorithm for Fuzzing-Based Grammar Inference

The comprehensive explanation of the grammar shown in [45, Figure 2.2.16] involves the process by which the start symbol expands into various symbols, namely numbers. The number symbol encompasses three distinct expansion rules: an integer symbol, an integer preceded by a plus sign, and an integer preceded by a minus sign. An integer symbol, in turn, comprises two expansion rules: either a single digit or a digit followed by another integer symbol, thus enabling recursion to represent integers with multiple digits. Essentially, a digit is a numerical value ranging from 0 to 9. Following these expansion rules systematically, valid integers can be generated according to the grammar structure. For instance, "-32", "007", and "+127" exemplify values created by adhering to these grammar rules. As the complexity of inputs increases, so does the complexity of the grammar. Crafting the grammar for intricate systems, such as a

## CHAPTER 2

programming language, from scratch without in-depth knowledge of that language would be a demanding task.

$$\begin{aligned}\langle \text{start} \rangle &::= \langle \text{number} \rangle \\ \langle \text{number} \rangle &::= \langle \text{integer} \rangle \mid + \langle \text{integer} \rangle \mid - \langle \text{integer} \rangle \\ \langle \text{integer} \rangle &::= \langle \text{digit} \rangle \mid \langle \text{digit} \rangle \langle \text{integer} \rangle \\ \langle \text{digit} \rangle &::= 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9\end{aligned}$$

**Figure 2.2.16:** Integer Grammar Example

### Fuzzing Techniques Summarization

Table 2.2.6 below provides a concise yet informative comparison of various fuzzing techniques based on their essential characteristics and effectiveness in detecting vulnerabilities.

**Table 2.2.6:** Comparison of different fuzzing techniques

<b>Fuzzing Technique</b>	<b>Vulnerability Detection</b>	<b>Strengths</b>	<b>Weaknesses</b>	<b>Adaptability to Large Systems</b>	<b>Accuracy</b>
<b>Mutation Fuzzing</b>	Buffer overflows, format string vulnerabilities	Requires minimal knowledge of the target program, good for surface-level issues	May miss complex logic flaws, limited in detecting intricate vulnerabilities	Moderate	Moderate
<b>Generation Fuzzing</b>	Protocol violations, semantic errors, logic flaws	Not reliant on existing samples, effective for complex vulnerabilities	May require substantial model/specification creation, and could miss known vulnerabilities	High	High
<b>File Format Fuzzing</b>	File format weaknesses	Identifies issues in how applications handle various file types	Requires knowledge of file formats, may not identify underlying system weaknesses	Moderate	Moderate
<b>Protocol Fuzzing</b>	Network-based vulnerabilities	Uncovers flaws in network-based software, effective for communication protocols	Challenging to create precise malformed packets, may not expose underlying logic issues	High	High

<b>White-box Fuzzing</b>	Complex vulnerabilities within software	Comprehensive visibility into the software, effectively targets specific paths/functions	Requires access to source code, time-consuming due to detailed analysis	High	High
<b>Gray-box Fuzzing</b>	Vulnerabilities in semi-restricted areas	Insights into the system's behavior, capable of discovering access control issues	Limited view of the system, may miss complex vulnerabilities in unexplored areas	Moderate	Moderate
<b>Black-box Fuzzing</b>	Common security issues	No internal knowledge required, efficient at finding surface-level vulnerabilities	Limited in finding intricate vulnerabilities, low code coverage, misses deeper system flaws	Low	Low
<b>DOM Fuzzing</b>	Browser's memory corruption or logic flaws	Manipulates web page elements, tests browser behavior with dynamic content	Challenges in reproducing crashes efficiently, identifying root cause of vulnerabilities	Low	Moderate
<b>API Fuzzing</b>	API functionality, security, data handling	Uncovers input validation, injection attacks, rate limiting issues	Requires knowledge of API endpoints, might not expose complex system issues	Moderate	Moderate
<b>Dumb Fuzzing</b>	Basic security vulnerabilities	Easy setup, reveals	Limited code coverage, misses intricate	Low	Low

		surface-level issues	vulnerabilities, relies on randomness		
<b>Smart Fuzzing</b>	Complex vulnerabilities within the software	Targets specific vulnerabilities, uncovers sophisticated flaws	Time-consuming to set up, requires in-depth understanding of the system	High	High
<b>Hybrid Fuzzing</b>	Multiple vulnerability classes	Combines strengths of different fuzzing methods, comprehensive coverage	Complexity in integration, might require considerable resources	High	High
<b>Coverage-guided Fuzzing</b>	Less-exercised code segments	Focuses on less-tested code areas, uncovers obscure vulnerabilities	Needs to prioritize untested areas, may overlook certain types of vulnerabilities	Moderate	Moderate
<b>Grammar-based Fuzzing</b>	Protocol, file format vulnerabilities	Detects issues related to invalid grammatical structures	Requires predefined grammars, may miss deeper logic flaws within the system	Low	Low

Fuzzing techniques can uncover certain vulnerabilities that other approaches, including some OSINT tools, might struggle to detect. Below are a few vulnerabilities that fuzzing methods can effectively target.

Fuzzing is particularly effective at uncovering **protocol-level vulnerabilities** that traditional security analysis tools might overlook. By subjecting network communication protocols such as HTTP, FTP, and SMTP to a series of malformed or unexpected inputs, fuzzing can reveal issues in how these protocols handle and interpret data. This might include detecting buffer overflows, command injection vulnerabilities, or denial-of-service conditions that arise from incorrect protocol handling [46]. Protocol fuzzing's ability to probe the details of data exchange and protocol implementation makes it a crucial technique for identifying flaws that could be exploited by attackers to compromise system integrity or disrupt services.

**Complex logic flaws** are challenging to detect with conventional testing methods, but fuzzing excels in this area [47]. Techniques like smart fuzzing and white-box fuzzing provide a deep dive into an application's internal logic, exploring various functional paths and interactions within the software. By testing how the application processes inputs and executes complex logical operations, these fuzzing methods can uncover intricate errors such as unauthorized access, privilege escalation, or data corruption. This thorough examination helps identify vulnerabilities in the application's design or functionality that might otherwise go unnoticed in less comprehensive testing environments.

**Memory corruption vulnerabilities**, including buffer overflows, memory leaks, and use-after-free errors, are effectively targeted by fuzzing techniques. Mutation and grammar-based fuzzing involve generating a wide range of malformed or unexpected inputs to stress-test the application's memory management routines. This rigorous approach can lead to crashes or unexpected behaviors caused by corrupted memory states [48]. By pushing the limits of how the application handles memory, fuzzing can

reveal critical issues that compromise the application's stability and security, often leading to exploitable conditions that traditional testing methods might miss.

Fuzzing is also proficient at uncovering vulnerabilities related to the **handling of unusual or malformed input data** [49]. Techniques such as file format fuzzing and grammar-based fuzzing are designed to test how applications manage and process input data that deviates from standard formats or syntax. This includes identifying parsing errors, input validation flaws, or crashes caused by data that the application is not equipped to handle properly. By examining how the application deals with non-standard or syntactically incorrect inputs, fuzzing can uncover vulnerabilities related to input validation and data processing that are crucial for ensuring robust software security.

In the context of web applications, DOM fuzzing is specialized in detecting vulnerabilities related to the **manipulation of web page elements using JavaScript**. This technique focuses on how browsers interpret and render dynamic content, potentially revealing issues such as memory corruption or logic errors within the browser's rendering engine. By targeting the DOM and examining the interactions between JavaScript and web page elements, DOM fuzzing can identify critical flaws that might not be evident through other testing methods [50]. This capability is particularly valuable for ensuring the security of web applications and protecting against vulnerabilities that could be exploited through client-side scripts.



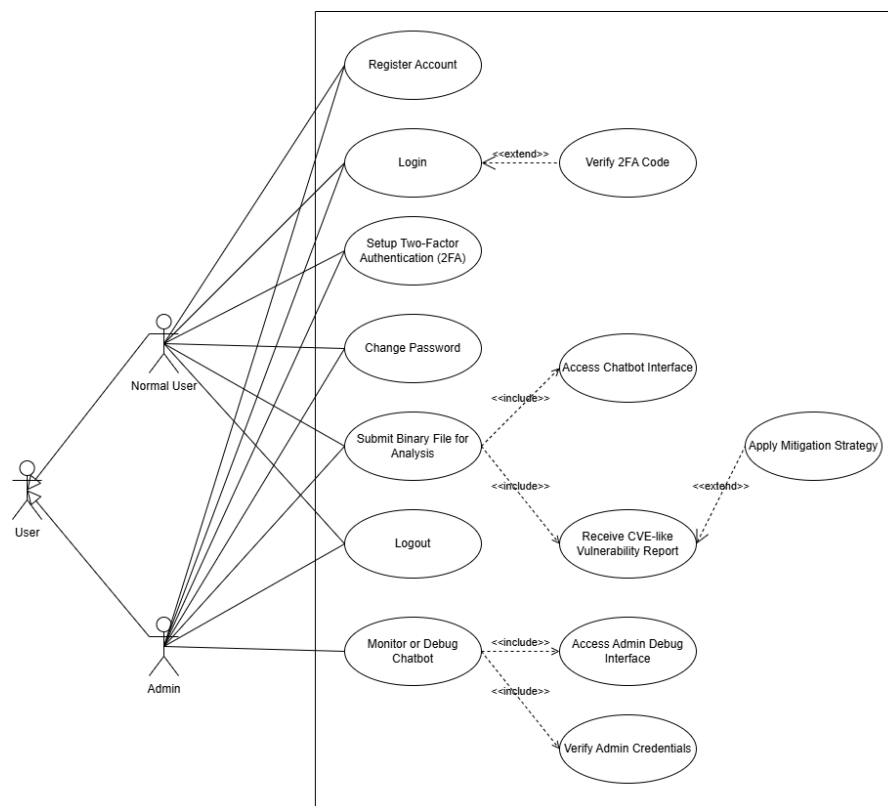
## CHAPTER 3

### System Methodology/Approach

This chapter outlines the methodology and structured approach adopted for developing the automated vulnerability assessment system. The system leverages the power of two state-of-the-art fuzzing engines, AFL++ [51] and Honggfuzz [52], which are integrated into a unified framework to enhance coverage and vulnerability detection efficiency. The methodology is presented through multiple diagrams that illustrate the overall system architecture, data flow, and internal logic. Additionally, functional and behavioral models such as use case diagrams and activity flows are provided to explain the user interactions and system processes involved in detecting and processing software vulnerabilities.

#### 3.1 Conceptual Design

##### 3.1.1 Use Case Diagram and Descriptions



**Figure 3.1.1:** Use Case Diagram of Multi-Fuzzer Platform for Automated Vulnerability Assessments

This section presents the use case model for the **Multi-Fuzzer Techniques for Automated Vulnerabilities Assessments** system. The use case diagram illustrates the interactions between the system and its users, along with the functionalities provided to different types of users. It identifies the system's primary actors, their associated use cases, and the relationships between them, including <<include>>, <<extend>>, and generalization relationships.

### Actors

- **User**  
This is a generalized actor that encompasses all users who interact with the system.
- **Normal User**  
A typical system user who interacts with the chatbot platform by registering an account, uploading binary files for analysis, and receiving vulnerability reports with suggested mitigation strategies.
- **Admin**  
A specialized user role that inherits all functionalities of a Normal User. In addition to standard capabilities, Admin users are granted access to advanced debugging and monitoring interfaces for system maintenance purposes.

The generalization relationship from Admin to Normal User indicates that Admin users have full access to all functionalities available to Normal Users, along with extended administrative privileges.

### Use Cases for Normal and Admin Users

- **Register Account**  
Enables new users to create an account before accessing the system functionalities.
- **Login**  
Authenticates existing users to allow system access.

- <<extend>> **Verify 2FA Code:** This optional use case is triggered when a user has enabled Two-Factor Authentication (2FA). The system prompts for a time-based verification code during login.
- **Setup Two-Factor Authentication (2FA)**  
Allows users to bind their account with a Time-Based One-Time Password (TOTP) authentication application for enhanced security.
- **Change Password**  
Permits users to change their existing password via the system settings.
- **Submit Binary File for Analysis**  
Allows users to upload a binary file to be assessed by the automated multi-fuzzer engine.
  - <<include>> **Access Chatbot Interface:** A necessary interaction where users engage with the chatbot responsible for processing the uploaded binary.
  - <<include>> **Receive CVE-like Vulnerability Report:** The chatbot returns a detailed vulnerability report based on the analysis of the submitted binary.
    - <<extend>> **Apply Mitigation Strategy:** Users have the option to apply system-recommended mitigation strategies to address the detected vulnerabilities.
- **Logout**  
Allows users to securely terminate their session.

### Additional Use Cases for Admin Users

- **Monitor or Debug Chatbot**  
Grants administrators the ability to oversee and troubleshoot the chatbot system.
  - <<include>> **Access Admin Debug Interface:** Provides access to the administrative debug interface containing advanced tools and logs.

- <<extend>> **Verify Admin Credentials:** Adds a secondary authentication layer before granting access to the debugging interface for security purposes.

### Use Case Relationships

- Generalization

The Admin actor is a generalization of the Normal User actor. This implies that Admins inherit all use cases associated with Normal Users.

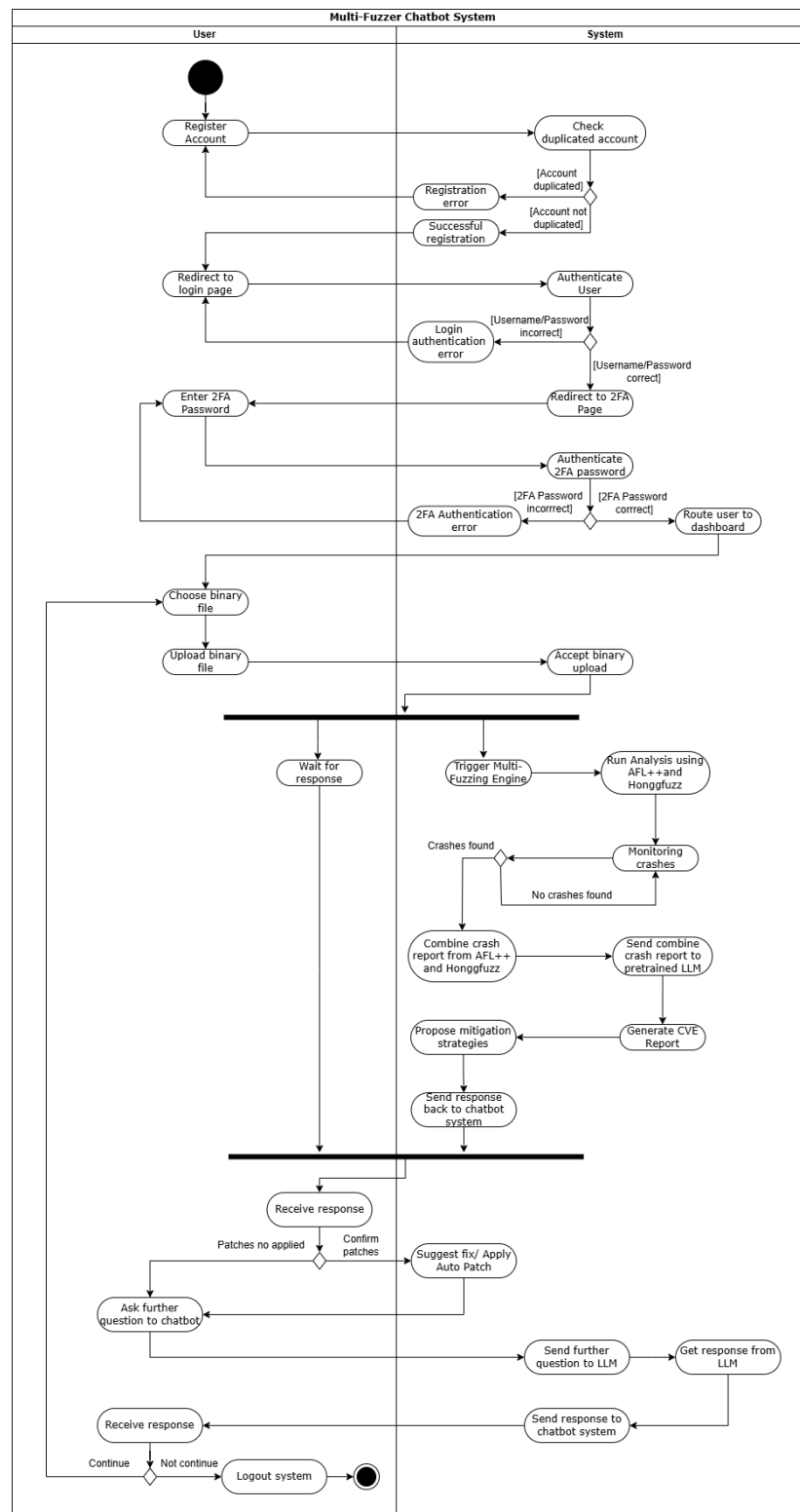
- <<include>>

Represents use cases that are essential components of a higher-level process. For instance, accessing the chatbot and receiving the CVE-like report are integral parts of submitting a binary for analysis.

- <<extend>>

Represents optional behaviors or conditional steps based on user configuration or system status. Examples include 2FA verification during login and the application of mitigation strategies post-analysis.

## 3.1.2 Activity Diagram



**Figure 3.1.2:** Activity Diagram of Multi-Fuzzer Platform for Automated Vulnerability Assessments

The activity diagram of the **Multi-Fuzzer Chatbot System** depicts the sequential flow of operations and the interactions between the **user** and the **system**, beginning from account registration to binary file analysis, vulnerability report generation, and the suggestion of mitigation strategies. The diagram is organized using two main swimlanes, **User** and **System** which effectively delineate the responsibilities and actions taken by each party throughout the process.

### Process Flow Description

#### **1. User Registration and Authentication**

The process begins with the user initiating the registration of a new account. The system checks for username duplication during this step. If a conflict is detected, the system returns an error and prompts the user to choose a different username. Upon successful registration, the user is redirected to the login page.

The login process requires the user to enter valid credentials. If **Two-Factor Authentication (2FA)** is enabled, the user is prompted to provide a time-sensitive verification code generated by a linked authenticator application (e.g., Google Authenticator). The system then validates both the credentials and the 2FA code. Failure in either step results in appropriate error messages. Upon successful verification, access to the user dashboard is granted.

#### **2. Binary Submission and Analysis**

Once authenticated, the user can upload a binary executable file for vulnerability assessment. The system processes the uploaded binary by launching the **multi-fuzzing engine**, which integrates fuzzers such as **AFL++** and **Honggfuzz** to execute the file under diverse input conditions, aiming to uncover any runtime anomalies or crashes.

#### **3. Crash Detection and Vulnerability Reporting**

During the fuzzing process, the system actively monitors for application crashes or abnormal behavior. If crashes are identified, detailed logs and metadata are compiled

into a **consolidated crash report**. This report is subsequently forwarded to a **pretrained Large Language Model (LLM)**, which interprets the raw data and transforms it into a **CVE-like vulnerability report**. This report outlines the nature of the vulnerability, its potential impact, and the relevant binary sections affected.

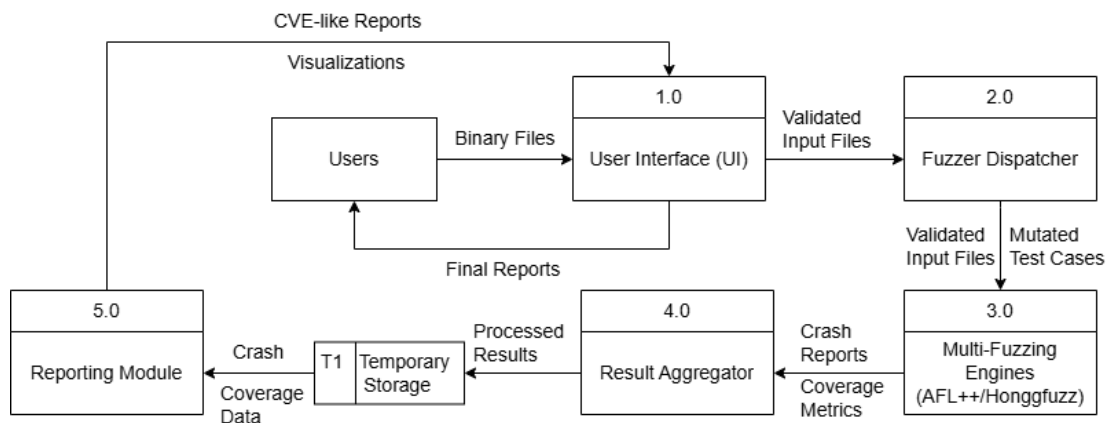
#### 4. Mitigation Strategy Generation and Interaction

Along with the generated report, the system provides a set of **automated mitigation strategies** tailored to the detected vulnerabilities. The user has the option to apply a suggested fix via the **auto-patching** function or engage further with the system's **chatbot interface**. Through natural language interaction, users may seek clarification, request alternative recommendations, or better understand the technical implications of the vulnerability. These interactions are continuously processed by the LLM to maintain conversational relevance and accuracy.

#### 5. Iteration and Session Termination

After receiving the vulnerability report and applying or reviewing mitigation strategies, the user can choose to either continue with another binary analysis or exit the platform by initiating the logout process. This loop allows for multiple cycles of vulnerability assessment, enhancing the system's utility in real-world testing workflows.

### 3.1.3 Data Flow Diagram



**Figure 3.1.3:** Data Flow Diagram of Multi-Fuzzer Platform for Automated Vulnerability Assessments

The Data Flow Diagram (DFD) for the fuzzing system outlines the logical flow of data across various functional components, starting with user interaction and ending with the generation of vulnerability reports. The process begins with the **User**, who interacts with the system via the **User Interface (UI)**. Through this interface, the user uploads binary files. This binary file input is then validated and forwarded to the **Fuzzer Dispatcher**, which acts as the central coordinator for the fuzzing tasks.

Upon receiving the input data, the Fuzzer Dispatcher performs mutation of the test cases. It then schedules and distributes these mutated inputs and binary file to two parallel fuzzing engines, **AFL++** and **Honggfuzz**. Both engines independently execute fuzzing tasks, generating outputs that include crash reports, log files, and code coverage information. These outputs are transmitted to the **Result Aggregator**, which serves as the system's analysis and deduplication unit.

The Result Aggregator consolidates the results from both fuzzers, eliminates duplicate crash data, and structures the remaining outputs for further analysis. This organized data is temporarily stored and passed to the **Reporting Module**. The Reporting Module utilizes the preprocessed data to generate CVE-like vulnerability descriptions and



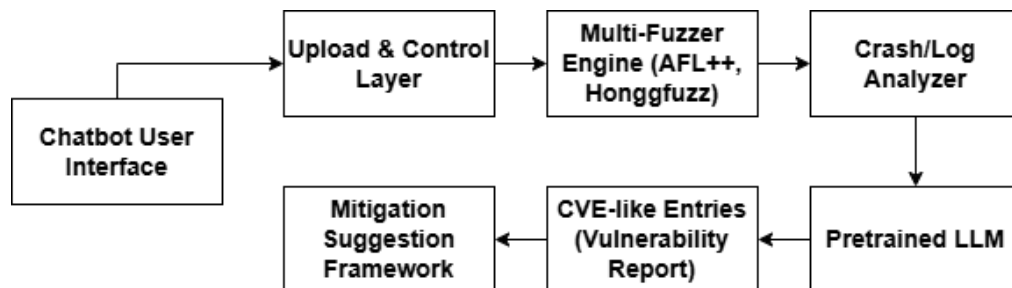
## CHAPTER 3

visual representations. These reports summarize the nature, location, and potential mitigation of detected vulnerabilities.

Finally, the processed outputs are returned to the **User Interface**, where they are displayed for the user in an accessible and structured format. The user can then review the complete assessment results, including technical summaries and recommended fixes, all derived from the initial uploaded binary.

## 3.2 Implementation Design

### 3.2.1 System Architecture Diagram



**Figure 3.2.1:** System Architecture Diagram of Multi-Fuzzer Platform for Automated Vulnerability Assessments

The architecture of the proposed system is designed to streamline and automate the vulnerability assessment process by integrating multiple advanced fuzzing tools within an interconnected framework. The system consists of several interconnected components that work together to analyze user-submitted software binaries, detect potential vulnerabilities, and produce detailed vulnerability reports. The architecture promotes modularity, automation, and scalability, allowing for efficient detection and interpretation of software flaws.

The process begins with the **Chatbot User Interface**, which serves as the entry point for user interaction. Through this interface, users can upload software binaries and configure fuzzing parameters in a user-friendly manner. The chatbot interface abstracts the complexity of underlying operations, making the system accessible even to users with limited technical expertise.

Once the input is received, the **Upload & Control Layer** is responsible for managing the session and directing the software binary to the appropriate processing modules. This layer ensures the correct handling of user input and initiates the fuzzing process by coordinating with the multi-fuzzer engine.

At the core of the system lies the **Multi-Fuzzer Engine**, which includes the integration of two complementary fuzzers, AFL++ and Honggfuzz. AFL++ employs coverage-guided fuzzing to explore diverse execution paths, while Honggfuzz applies aggressive

mutation techniques to stress-test the application under varying conditions. Running both fuzzers in parallel increases the breadth and depth of vulnerability discovery, improving overall assessment effectiveness.

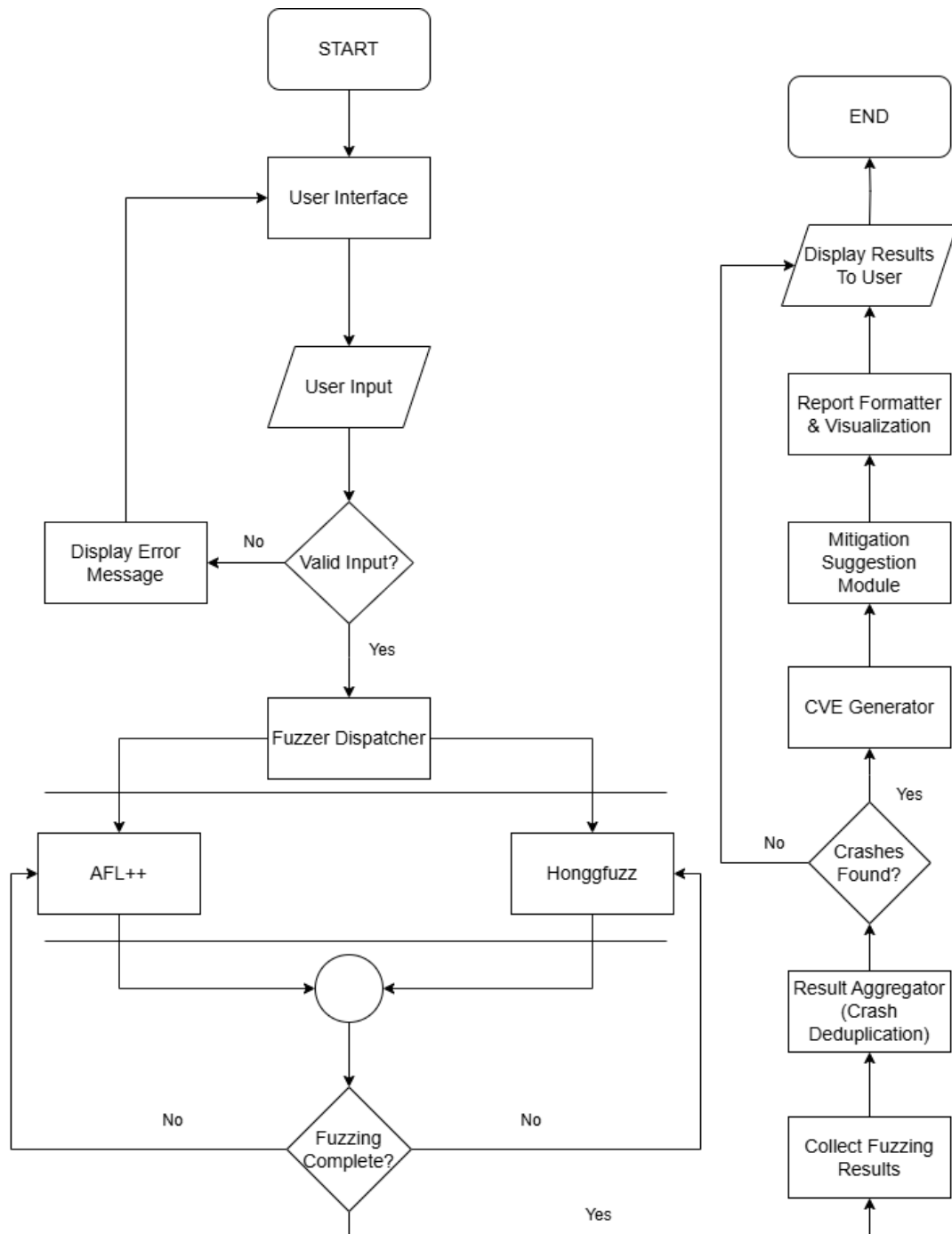
The results from the fuzzers are then analyzed by the **Crash/Log Analyzer**, which captures and filters crash data and execution logs. This component organizes the data to identify anomalies or signs of software bugs, preparing it for further interpretation by the next module in the pipeline.

Following analysis, the structured crash data is processed by a **Pretrained Large Language Model (LLM)**. This module leverages natural language understanding capabilities to interpret the context of the detected bugs and generate descriptive insights into their nature. It translates low-level fuzzing outputs into understandable descriptions that capture potential risks and security implications.

These descriptions are compiled into structured outputs by the **CVE-like Entries (Vulnerability Report)** module. The vulnerability report provides users with clear and actionable information, including the type of vulnerability, possible impact, and conditions under which the flaw may be exploited. While not integrated into an official CVE database, the structure mimics industry-standard vulnerability formats for familiarity and practical utility.

Finally, the **Mitigation Suggestion Framework** provides follow-up recommendations based on the nature of the discovered vulnerabilities. These suggestions may include code-level changes, configuration updates, or general secure development practices. By offering immediate guidance, the framework enhances the security posture of the software and reduces the window of exposure to potential threats.

## 3.2.2 Flowchart



**Figure 3.2.2:** System Flowchart of Multi-Fuzzer Platform for Automated Vulnerability Assessments

The flowchart of the proposed fuzzing system illustrates a comprehensive and structured sequence from the initial user interaction to the final report generation. The

process begins at the **Start node**, where the system is initialized, and the user begins their interaction. This marks the beginning of a vulnerability assessment cycle, which is designed to be user-friendly and highly automated.

The first operational step occurs at the **User Interface (UI)**. This front-end component provides an accessible platform for users to upload the binary files they wish to fuzz. The UI also allows users to configure essential fuzzing parameters, including the selection of fuzzing engines (AFL++ and/or Honggfuzz), mutation strategies, and the maximum runtime for each fuzzing session. To support a broad range of users, including those without deep technical backgrounds, the UI incorporates a **chatbot interaction panel** for guidance and result interpretation, as well as a results display area for visualizing final outputs.

Following user input, the system performs a validation check through the **"Valid Input?" decision node**. Here, the uploaded binaries and configuration parameters are examined for correctness and completeness. This validation step ensures that the binary file format is appropriate, all required fields are filled, and the settings align with system constraints. If the inputs are invalid, the system loops back to the UI and prompts the user to correct them. This feedback loop enforces input integrity and prevents unnecessary processing.

Once inputs are validated, control is passed to the **Fuzzer Dispatcher**, which acts as the core task dispatcher and input mutation engine. It intelligently schedules tasks across available fuzzers and applies mutation algorithms to generate diverse test cases based on the input seed corpus. The dispatcher ensures optimal workload distribution and prepares each fuzzing engine for execution.

The system then initiates **parallel fuzzing execution**, where both AFL++ and Honggfuzz engines run concurrently. The AFL++ engine focuses on coverage-guided fuzzing and employs evolutionary algorithms to discover unique execution paths. In

contrast, Honggfuzz offers enhanced instrumentation, fault injection, and persistent fuzzing techniques. Running both engines in cycle increases fuzzing depth and broadens the vulnerability detection spectrum.

During execution, the system continuously monitors the process through the **"Fuzzing Complete?" decision node**, which checks whether the defined runtime or iteration limits have been reached. If not, the fuzzers continue executing. If complete, the result router resumes its role by collecting output data. This includes crash logs, sanitizer outputs, and code coverage metrics, which are critical for vulnerability assessment.

The next phase involves the **Result Aggregator**, which processes raw fuzzing outputs. Its first function is crash deduplication, which eliminates redundant crash reports by comparing stack traces or crash signatures. It then formats the refined data into a structured format that is suitable for downstream analysis. This step is crucial in reducing noise and highlighting unique issues.

Subsequently, the system checks for crash presence via the **"Crashes Found?" decision node**. If crashes are detected, they are sent for vulnerability interpretation. If not, the process proceeds directly to reporting. In the case of valid crashes, the system invokes the **CVE-like Report Generator**, which leverages a pretrained large language model (LLM) to analyze crash characteristics and generate structured reports. These CVE-like reports identify the nature, scope, and impact of the vulnerabilities. In parallel, the **Mitigation Recommender** module suggests remediation strategies, such as input sanitization, logic refactoring, or memory protection techniques.

The final processing step is handled by the **Reporting and Visualization Module**, which formats the CVE reports and translates analytical data into visual elements. This includes charts, coverage graphs, and vulnerability heat maps that help users understand complex issues immediately. Although the system does not rely on a persistent database,

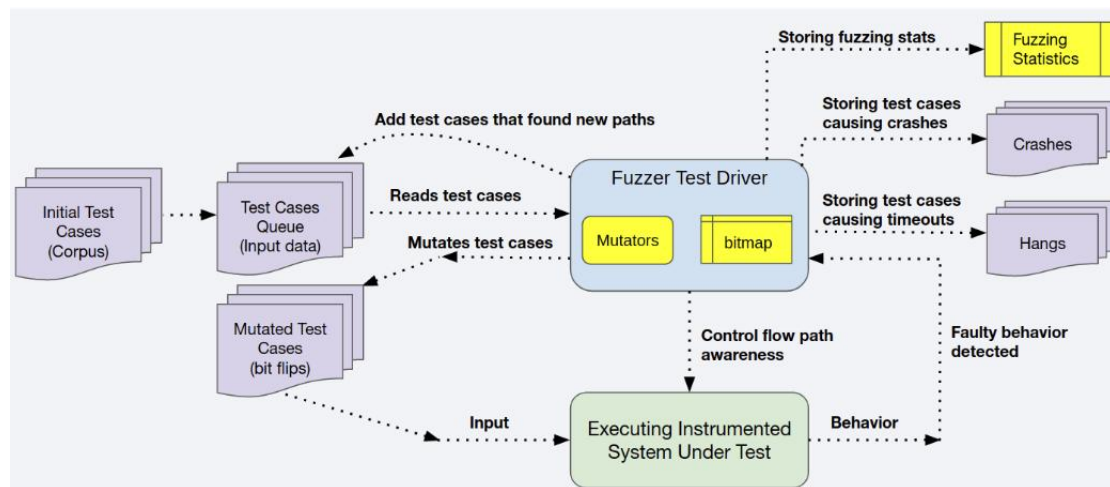
## CHAPTER 3

all reports and visualizations are generated in real time and presented directly to the user.

Ultimately, the results are shown in the "**Display Results to User**" stage via the UI, completing the loop from input to insight. Users can view CVE reports, crash data, and graphical summaries, enabling them to make informed security decisions. The process then reaches the **End node**, which signifies the conclusion of the fuzzing task. At this point, the user may choose to start a new assessment or terminate the session.

In conclusion, this flowchart embodies a streamlined yet powerful framework for conducting automated, multi-engine fuzz testing.

### 3.2.3 Internal Fuzzing Logic



**Figure 3.2.3:** Overview of an AFL++ fuzz testing session [53]

**AFL++** (American Fuzzy Lop Plus Plus) is a powerful and highly configurable fuzzer designed for testing software vulnerabilities through an instrumentation-guided approach. As shown in Figure 3.1, the AFL++ fuzzing process begins with the preparation of an initial set of test cases, known as the corpus. These test cases are stored in a directory and serve as the foundation for the fuzzing session. AFL++ utilizes a QEMU plugin to instrument the code under test, enabling it to monitor execution paths and collect detailed coverage information of the binaries.

During the fuzzing session, AFL++ employs internal and external mutators to generate a large number of new test cases from the initial corpus. Each of these new test cases is executed on the target software, and AFL++ monitors the execution for any signs of faulty behavior, such as crashes or hangs. Crashes are identified when the program exhibits abnormal termination due to invalid inputs, while hangs occur when the program becomes unresponsive. Test cases that lead to crashes are moved to a dedicated directory, while those that exceed a specified timeout are stored in the hangs directory for further investigation.



AFL++ maintains a bitmap that tracks code coverage by incrementing indices corresponding to executed instrumentation points. This bitmap helps the fuzzer to focus on exploring new execution paths. Test cases that introduce novel paths are prioritized for further mutation. The fuzzer's efficiency is reflected in various statistics, including the number of unique crashes detected and the overall execution speed. By continuously generating, mutating, and testing inputs, AFL++ effectively uncovers vulnerabilities and provides insights into the strength of the software under test.

Meanwhile, **Honggfuzz** is a versatile and powerful security-oriented fuzzer designed to uncover vulnerabilities in software by using both evolutionary and feedback-driven fuzzing techniques. It can operate in a multi-process and multi-threaded environment, allowing it to efficiently utilize all available CPU cores with a single instance. This scalability and performance optimization makes Honggfuzz particularly effective in performing extensive fuzzing sessions on a wide range of software targets. Additionally, Honggfuzz supports both software-based and hardware-based fuzzing modes, providing users with multiple strategies to enhance code coverage and uncover hidden bugs.

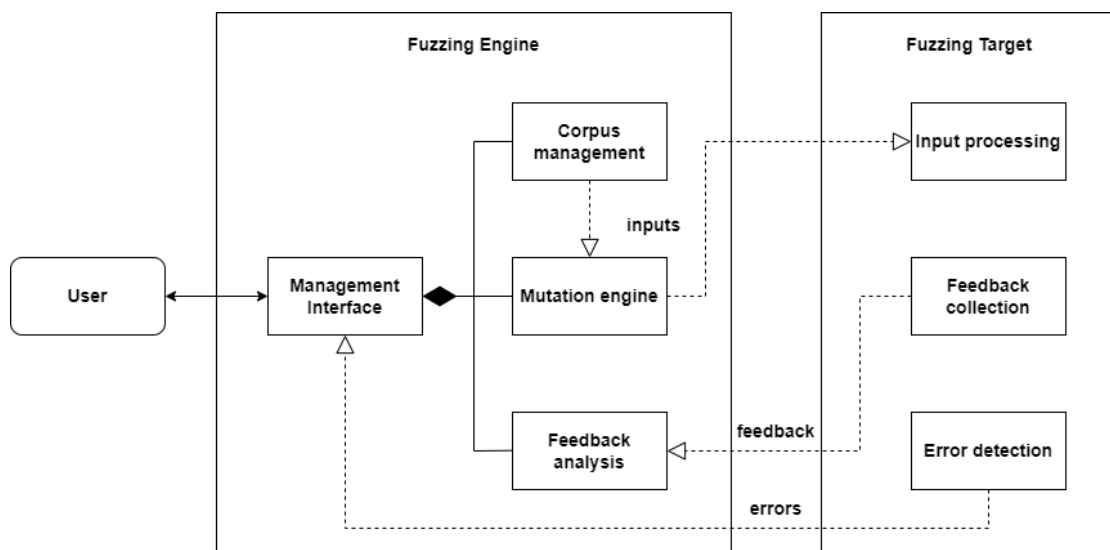
Honggfuzz detects crashes by monitoring the behavior of the target program as it processes a series of generated inputs. The fuzzer begins by feeding a set of initial test cases into the target application. This corpus can be a collection of sample inputs or even an empty directory for feedback-driven fuzzing. Honggfuzz modifies and mutates these inputs using various mutation strategies to create new test cases, which are then executed on the target program. As the program runs, Honggfuzz uses feedback from the execution to guide further mutations and generate additional inputs. This feedback mechanism allows Honggfuzz to prioritize test cases that explore new execution paths, increasing the chances of discovering vulnerabilities.

To monitor and detect crashes, Honggfuzz leverages low-level system interfaces such as `ptrace` on Linux and NetBSD, which allow it to intercept and examine the behavior of the target process during execution. This approach enables Honggfuzz to detect

abnormal behavior, such as crashes or hangs, that may indicate a security flaw. When a test case causes the program to crash or behave unexpectedly, Honggfuzz captures and logs the signal, along with the offending input, and stores this information in a dedicated directory for further analysis. This method ensures that even sensitive crashes or errors that might be suppressed or ignored by the application are detected and recorded.

Moreover, Honggfuzz supports a persistent fuzzing mode, where the target process remains active and repeatedly executes the fuzzed API. This mode allows for more efficient testing of functions or APIs that require continuous interaction to expose vulnerabilities. By maintaining the process state between iterations, Honggfuzz can achieve high iteration rates, significantly speeding up the fuzzing process. The fuzzer also includes a feature for automatic corpus management, which involves minimizing the corpus by removing redundant or ineffective test cases. This continuous refinement of the input set ensures that Honggfuzz remains focused on the most promising inputs, thereby maximizing the effectiveness of the fuzzing session.

### 3.2.4 Fuzzer Architecture



**Figure 3.2.4:** Coverage-Guided Fuzzer Architecture

Coverage-guided fuzzing, such as AFL++ and Honggfuzz employs a structured approach to detect crashes by utilizing a feedback-driven mechanism that iteratively refines test inputs based on the software's responses. The process begins with the fuzzer generating a range of test inputs, which are then applied to the target software. If these inputs cause a failure or crash, the fuzzer records the specific input data that triggered the issue. This feedback is crucial as it provides insights into how the software behaves under different conditions, enabling the fuzzer to learn from these interactions and adapt its testing strategy accordingly. This iterative feedback loop enhances the fuzzer's ability to uncover hidden and complex vulnerabilities that may not be immediately apparent through random testing methods.

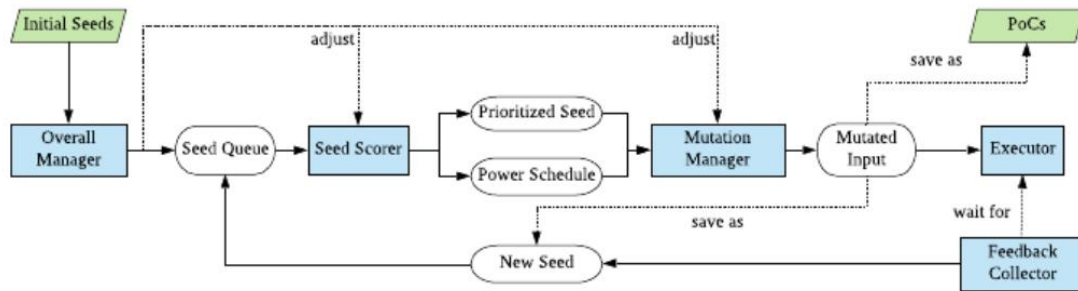
The architecture of guided fuzzing also includes sophisticated **corpus management** and a **mutation engine**. The seed corpus consists of initial, handcrafted inputs designed to cover various aspects of the target software, while the live corpus is continually updated with new, generated inputs during the fuzzing process. Mutators apply different strategies to these inputs, creating variations that help in exploring new paths and conditions within the software. By evolving and diversifying the inputs, coverage-guided fuzzing increases the likelihood of triggering crashes that could reveal

previously undiscovered vulnerabilities. This dynamic approach ensures that the fuzzer systematically explores different execution paths, maximizing the chances of identifying critical bugs.

**Feedback analysis** and **error detection** are central to the effectiveness of guided fuzzing. The fuzzer collects detailed data on the software's behavior in response to each input, including code coverage metrics, execution traces, and performance data. This feedback helps in determining whether an input has led to a significant error or crash. When a crash is detected, the fuzzer records comprehensive information about the incident, such as the input data, logs, and backtraces, providing valuable insights into the nature of the vulnerability. This detailed error detection process allows developers to identify the root causes of crashes and address them effectively.

Finally, the **management interface** of the guided fuzzing framework facilitates the coordination and monitoring of the fuzzing process. It enables users to manage and control various aspects of the testing, including executing individual runs, updating the corpus, and analyzing errors. The interface also supports the examination of detected errors and the processing of crash artifacts, ensuring that the fuzzing process is conducted efficiently and effectively.

### 3.2.5 Detailed Fuzzing Architecture in FOT



**Figure 3.2.5:** Detailed Fuzzing Architecture in FOT [54]

- The **Overall Manager** is the central component responsible for coordinating the multi-threaded parallel fuzzing operations. It manages the workload distribution among worker threads and handles the import of seed inputs from external sources, such as symbolic executors like KLEE. The manager's configurability allows users to select different management strategies, and its extensibility supports integration with various seed generation tools, enhancing the fuzzer's adaptability and scalability.
- The **Seed Scorer** plays a crucial role in prioritizing seeds and scheduling their mutation. It evaluates seeds based on a selected scoring strategy and decides how many new inputs should be generated from each seed. This component's configurability allows users to choose from several built-in scoring methods or develop custom strategies, while its extensibility enables the implementation of personalized seed scoring approaches.
- The **Mutation Manager** handles the application of mutation operators to the selected seeds. It supports both random and predefined grammar-based mutations. This component is configurable, providing a range of mutation operators for users to select according to their needs. It is also extensible, allowing users to add their own mutation operators to tailor the fuzzing process to specific requirements.
- The **Executor** is responsible for running the Program Under Test (PUT). It offers configurability, such as the option to enable or disable the use of forking, and can be extended to accommodate different testing scenarios. For example, users can add secondary executors for differential testing of multiple PUTs.

## CHAPTER 4

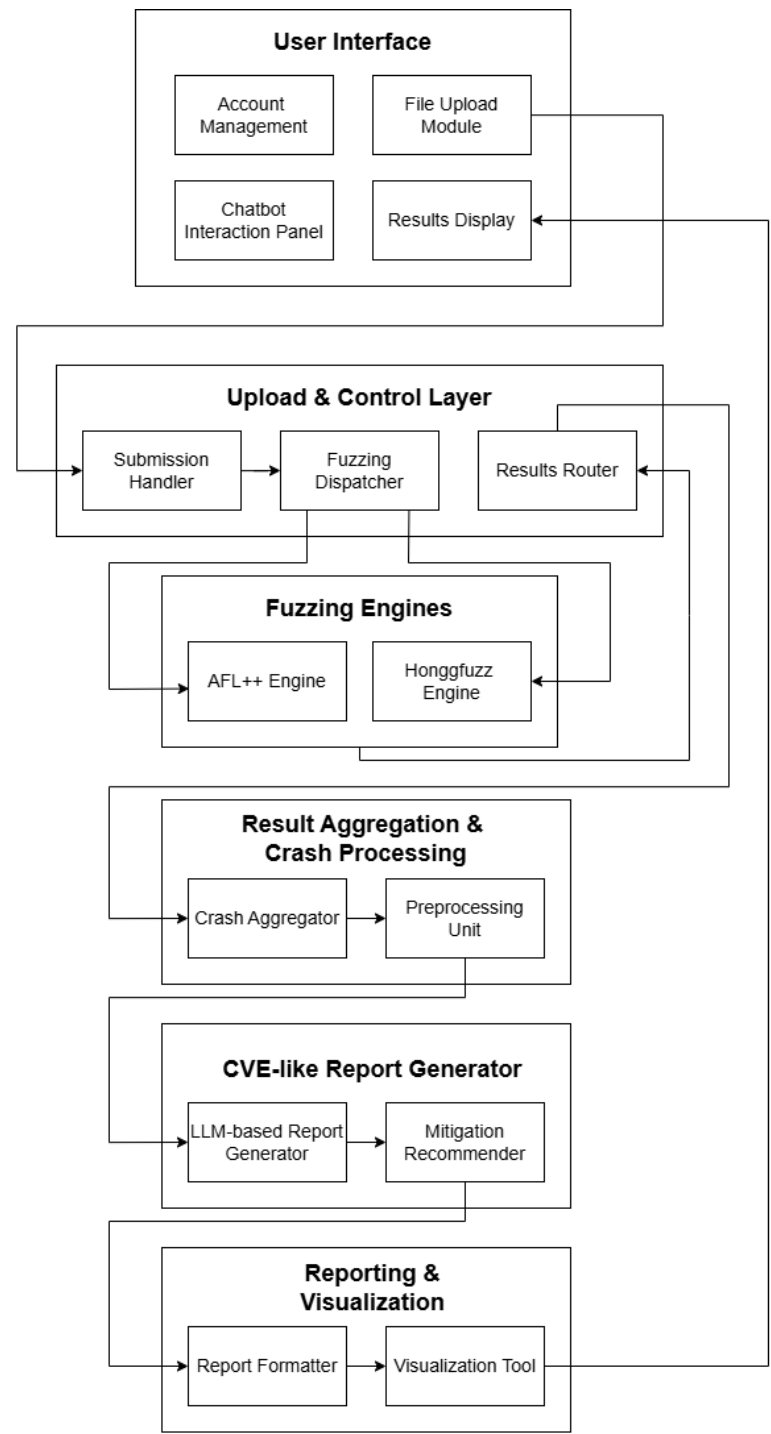
### System Design

This chapter details the structural design and core components of the proposed automated vulnerability assessment system. It begins with a high-level block diagram that outlines the major functional modules and their interactions. Each system component, ranging from the user interface and fuzzing dispatcher to the fuzzing engines and result aggregator, is described in terms of its role and technical specifications. The chapter also covers the integration of a large language model (LLM) for CVE-style report generation, explaining how crash data is transformed into structured vulnerability descriptions. Finally, component interaction and data operation flows are presented to illustrate how information moves through the system, enabling seamless vulnerability detection, reporting, and mitigation.

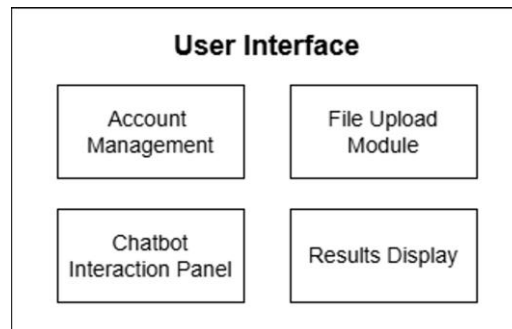
#### 4.1 System Block Diagram

The system block diagram serves as a top-down architectural overview of the Multi-Fuzzer Chatbot System. This diagram illustrates the structural and functional relationships between key modules, starting from the user interface to the core multi-fuzzing logic, result analysis, and final reporting. The system is organized into six logical layers, each dedicated to a specific aspect of the workflow. These layers operate sequentially, with data flowing from top to bottom and feedback or output rendered back to the user.

Each block in the diagram is designed with modularity and scalability in mind, allowing for clear separation of concerns and streamlined integration of future improvements. The description below details the role of each block and the data flow between them.



**Figure 4.1.1:** Overall Block Diagram of Multi-Fuzzer Platform for Automated Vulnerability Assessments



**Figure 4.1.2:** User Interface Block Diagram

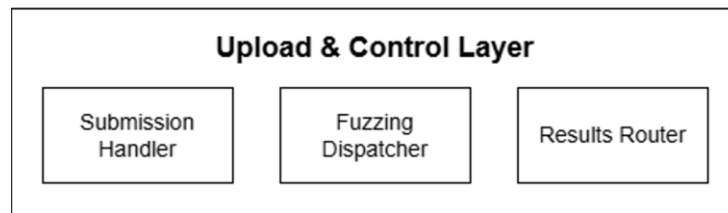
The **User Interface (UI) Block** is the front-facing gateway for users to interact with the fuzzing system. It is designed with accessibility and usability in mind, integrating multiple modules that together streamline the end-user experience:

- **Account Management:** This component governs user registration, authentication, and security protocols, including two-factor authentication (2FA). It ensures that only verified users can access the system, protecting sensitive data and operations.
- **File Upload Module:** This module allows users to upload binary executables or other software artifacts for vulnerability testing. Uploaded files are subjected to initial client-side validation before being handed off to the processing pipeline.
- **Chatbot Interaction Panel:** This interactive console integrates an AI assistant (powered by a pretrained LLM), enabling users to query real-time progress, request explanations of vulnerabilities, and receive guidance on configuration or mitigation.
- **Results Display:** This module is responsible for rendering output in both textual and visual formats. It receives structured data from downstream blocks and presents CVE-like reports, charts, and actionable suggestions in a digestible format.

### Functional Role:

This layer captures all user-driven events and input. It serves as both the starting point and the final feedback terminal in the system workflow. Commands and data collected here are transmitted to the processing engine for fuzzing and analysis.





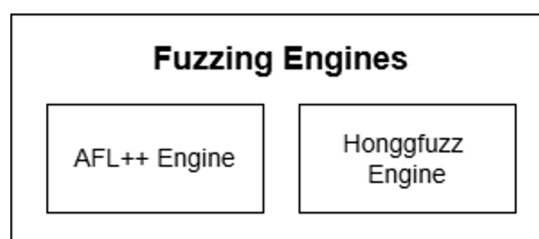
**Figure 4.1.3:** Upload & Control Layer Block Diagram

The **Upload & Control Layer** functions as the system's middleware, bridging frontend inputs with backend operations. It handles task preparation, validation, and control logic:

- **Submission Handler:** Validates uploaded binaries for correctness, format, and compatibility with selected fuzzing engines. It also generates metadata associated with each file (e.g., filename, size, upload timestamp) to aid in processing.
- **Fuzzing Dispatcher:** Based on the user's configuration (engine selection, time constraints, mutation strategies), this component initializes fuzzing jobs. It allocates resources and schedules execution threads to AFL++ and/or Honggfuzz engines in parallel.
- **Results Router:** Gathers output files, crash logs, and metadata from the fuzzers. It ensures that each data packet is correctly channeled to the aggregation and analysis layers downstream.

#### Functional Role:

This layer ensures operational coherence between the UI and core fuzzing logic. It is critical for managing task lifecycles, orchestrating engine behavior, and routing data without loss or misconfiguration.



**Figure 4.1.4:** Fuzzing Engines Block Diagram

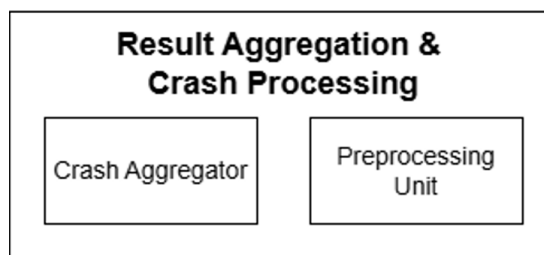
The **Fuzzing Engines Block** is the core computational unit responsible for vulnerability discovery through dynamic testing. It supports parallel execution of different fuzzers:

- **AFL++ Engine:** A highly optimized coverage-guided fuzzer. It uses genetic algorithms to mutate inputs and detect diverse execution paths that may contain vulnerabilities such as buffer overflows or memory corruption.
- **Honggfuzz Engine:** A complementary fuzzer that supports both black-box and white-box fuzzing. It includes features such as persistent mode, software-based fault injection, and feedback-based instrumentation.

Each engine is provided with mutated test cases from the dispatcher. They run these inputs in sandboxed environments, collecting metrics like instruction coverage, edge coverage, crash signals, and execution time.

#### **Functional Role:**

This layer performs intensive computation to identify software vulnerabilities by executing thousands of mutated inputs and observing the target program's behavior. The outputs here are crash data and coverage logs forwarded to the next layer for aggregation.



**Figure 4.1.5:** Result Aggregation and Crash Processing Block Diagram

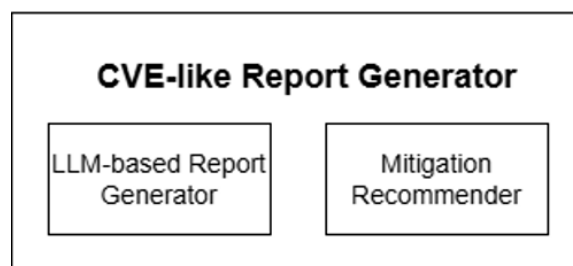
The **Result Aggregation and Crash Processing Block** is responsible for collecting, deduplicating, and structuring fuzzing results from the engines. It acts as a filtration and formatting step before deep analysis:

- **Crash Aggregator:** Receives crash reports from both fuzzers and removes duplicate crashes caused by the same bug. It groups crashes by signal, stack trace similarity, and memory access patterns to avoid redundant analysis.

- **Preprocessing Unit:** Converts aggregated crash data into a standardized format suitable for AI analysis. This may include extracting instruction pointers, memory addresses, crash logs, and affected function names.

#### Functional Role:

This layer reduces noise and computational overhead by filtering unnecessary or duplicate crash entries. It produces structured and meaningful datasets that serve as input to the vulnerability interpretation engine.



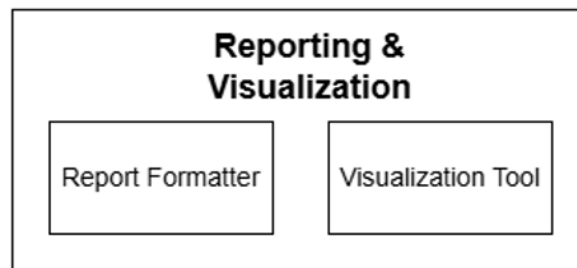
**Figure 4.1.6:** CVE-like Report Generation Block Diagram

The **CVE-like Report Generation Block** interprets crash data and automatically transforms it into actionable vulnerability reports using machine learning models:

- **LLM-based Report Generator:** Utilizes a pretrained Large Language Model (LLM) fine-tuned for cybersecurity tasks. It analyzes crash data and synthesizes reports in a format similar to industry-standard CVE entries, including fields like vulnerability type, severity, affected component, and possible impact.
- **Mitigation Recommender:** Builds on the LLM output to suggest relevant fixes. These may include input validation, memory access bounds enforcement, patching references, or static code instrumentation techniques.

#### Functional Role:

This block provides context and insights. It transforms low-level machine crashes into high-level descriptions and remediation strategies, helping users understand and address software vulnerabilities quickly.



**Figure 4.1.7:** Reporting and Visualization Block Diagram

The **Reporting and Visualization Block** is the system's final output interface, turning analysis into visually interpretable formats:

- **Report Formatter:** Takes the LLM-generated CVE-like entries and formats them into coherent technical documentation. It ensures clarity, sectioning (e.g., Summary, Root Cause, Impact, Suggested Fixes), and readability.
- **Visualization Tool:** Renders interactive charts, bar graphs, heatmaps, and trend lines to represent key statistics such as fuzzing coverage, number of unique crashes, and mitigation effectiveness. It enhances understanding for both technical and non-technical users.

Despite the absence of persistent storage, this block leverages in-memory data structures to display reports and visuals in real time. The outputs are routed back to the Results Display section of the UI, completing the feedback loop.

#### **Functional Role:**

This layer maximizes usability and interpretability of complex results. It bridges technical output and user experience, empowering decision-making through clarity and interaction.

### 4.2 System Components Specifications

This section presents the technical specifications and key components that form the foundation of the developed fuzzing-based automated vulnerability assessment system. Each module in the architecture is carefully selected and configured to ensure efficient fuzzing, crash analysis, and vulnerability reporting.

#### AFL++ and Honggfuzz

The system utilizes two prominent fuzzing engines, **AFL++** and **Honggfuzz**, to maximize code coverage and increase the likelihood of discovering runtime vulnerabilities. Both tools are deployed in a virtualized environment running Ubuntu 22.04, enabling compatibility, reproducibility, and sandboxed testing.

**AFL++** is operated in QEMU mode, which allows for binary-only fuzzing without requiring source code instrumentation. This is particularly beneficial when handling third-party or closed-source binaries. **Honggfuzz** is configured to operate through standard input, offering an efficient method of feeding test cases directly into the target application.

To enhance fuzzing accuracy and depth, both engines rely on a combination of custom and general seed files. Custom seeds are designed for specific binary formats or edge cases, while general seeds provide broader test coverage. The fuzzers use their native input mutation algorithms to automatically generate variations of the seeds and attempt to trigger anomalous behaviors, including memory corruption, crashes, and logic errors.

#### Fuzzing Dispatcher

The Fuzzing Dispatcher acts as the operational brain of the system, orchestrating the fuzzing tasks and managing system resources. When a user initiates a fuzzing session through the interface, the dispatcher redirects the selected binary to an execution environment where either or both fuzzers are launched. To optimize performance within the resource constraints of the virtual machine, the dispatcher limits simultaneous sessions to a maximum of four concurrent fuzzing jobs.

As binaries are submitted, the dispatcher manages their execution based on resource availability. Users are allowed to open multiple tabs to initiate fuzzing jobs, which are executed in parallel up to the defined system limit. Input selection and mutation are entirely handled by the respective fuzzers, ensuring minimal overhead and tight integration with each tool's internal logic.

### Result Aggregator

Crash detection and reporting are handled by the Result Aggregator, which consolidates outputs from both AFL++ and Honggfuzz. When a crash is detected, AFL++ logs are parsed to extract crash signatures, while Honggfuzz's output is reviewed by capturing the last 30 lines of its continuously updated report file—where recent crashes are appended.

The aggregator merges the extracted information into a **single, structured text file**, which is saved in a predefined crash directory on the system. To maintain focus on the most recent findings, previous crash reports are automatically removed once new data is stored. Each aggregated crash report is then converted into a structured **JSON format**, capturing relevant technical details and context. This JSON file becomes the input for the downstream LLM analysis.

### LLM Engine

At the core of the vulnerability interpretation workflow is a Large Language Model (LLM) powered by **OpenAI's GPT-3.5 Turbo**. This model is accessed through a custom-built PHP interface and receives the structured JSON crash data alongside a tailored prompt. The prompt instructs the LLM to:

- Interpret the technical details of the crash.
- Generate a CVE-style report that classifies the vulnerability.
- Recommend possible mitigation strategies.

In addition, the system maintains a predefined mapping of known vulnerabilities to Bash-based mitigation scripts. If the LLM identifies a vulnerability and suggests a

matching keyword, a **mitigation button** is automatically generated in the user interface. When clicked, this triggers the relevant script to apply the recommended fix directly to the system.

### Temporary File-Based Storage

Instead of using a traditional database, the system employs file-based storage for simplicity and speed. All crash reports, logs, and LLM inputs/outputs are stored as structured text files in organized directories. This lightweight approach is sufficient for the scope of the project and ensures that crash data is always accessible in a readable, traceable format.

The storage logic is designed to maintain only the latest valid crash report for each session, avoiding clutter and confusion. This also ensures that the LLM always analyzes the most up-to-date information.

### Visualization and User Feedback

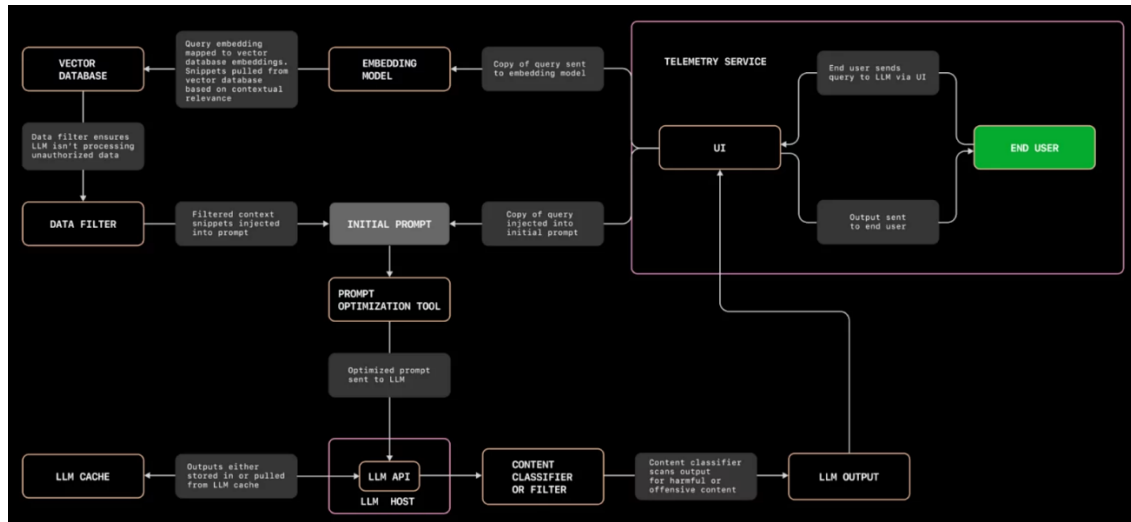
User feedback is delivered through a chatbot-style interface developed using PHP and JavaScript. Once the LLM processes a crash report, it presents the vulnerability summary and proposed mitigations via conversational text. If a recommended mitigation is available, a button is shown to allow immediate remediation through an underlying Bash script.

The outcome of these scripts is presented using **structured before-and-after comparisons**. For example, in the case of a disk space exhaustion vulnerability, the chatbot may report a reduction in disk usage from 99% to 60% post-mitigation. This form of direct visual confirmation helps users quickly understand the impact and success of the applied solution.

While the system currently focuses on textual visualizations, it is designed with flexibility for future integration of interactive charts or graphs to enhance reporting.

### 4.3 LLM Architecture and CVE Generation Design

#### 4.3.1 LLM architecture diagram



**Figure 4.3.1:** Emerging Architecture for LLM Applications [55]

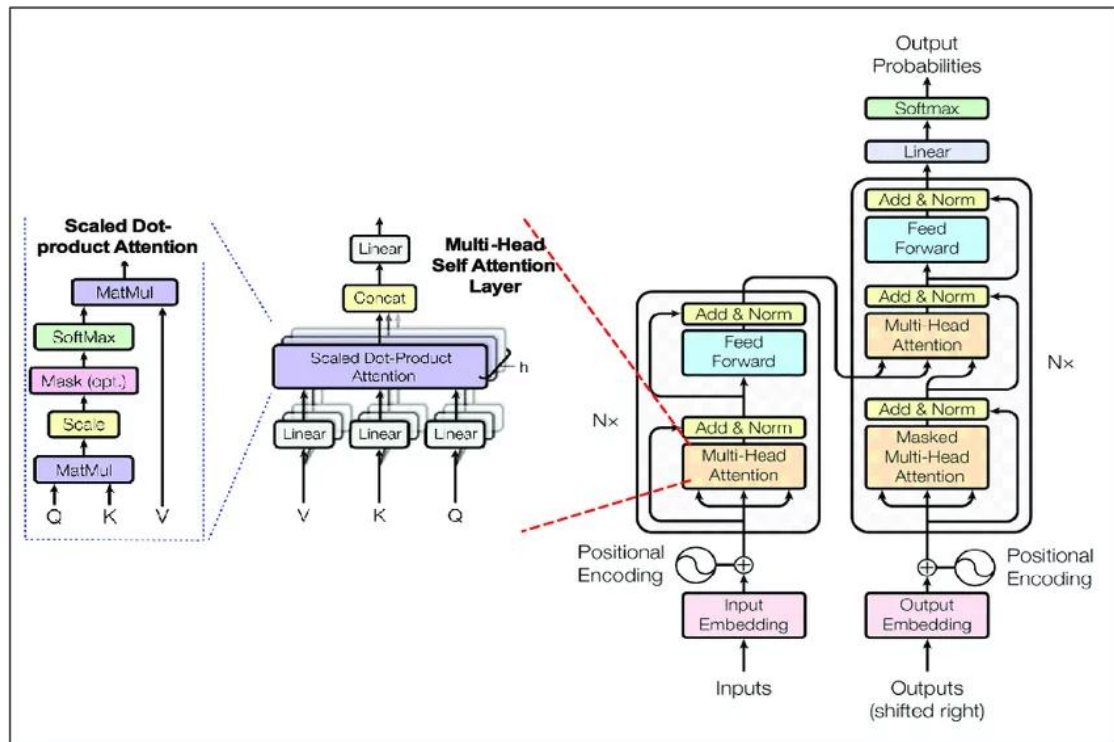
The architecture of Large Language Models (LLMs) is a complex and multilayered structure, designed to process and generate text with remarkable accuracy and fluency. At its core, an LLM consists of several key layers, each playing a specific role in handling and transforming the input data. The process begins with **embedding layers**, where the input text is converted into numerical representations called embeddings. These embeddings capture the semantic meaning of words, allowing the model to operate on numerical data rather than raw text. However, since LLMs must understand not just the words themselves but also the order in which they appear, **positional encoding** is applied. This technique assigns position-based values to each token, helping the model recognize the sequential nature of the text.

The architecture further includes **attention layers**, particularly the self-attention mechanism, which allows the model to focus on the most relevant parts of the input sequence. By weighing the importance of different words based on their context, the model can capture intricate relationships between words, regardless of their position in the sentence. Following the attention mechanism, the data passes through **feedforward layers** neural networks that further refine the model's understanding by adding depth and complexity to the representations. Finally, the **output layer** generates the final



predictions or text, completing the transformation from raw input to meaningful language output. This layered approach enables LLMs to handle a wide range of language processing tasks, from text generation to translation and summarization.

### 4.3.2 Transformer Architecture



**Figure 4.3.2:** Transformer Architecture [55]

The transformer architecture is a pivotal advancement in the field of language processing, particularly in the development of LLMs. Introduced in 2017, transformers revolutionized how models handle sequential data by relying solely on self-attention mechanisms, eliminating the need for recurrent neural networks. The process begins with **input embeddings**, where the text is converted into numerical data, followed by **positional encoding** to maintain the order of words in a sentence. The most significant feature of transformers is the **multi-head self-attention** mechanism. This allows the model to analyze different parts of the input text simultaneously, capturing various relationships and traces within the sequence. Each attention head focuses on different aspects of the input, and their outputs are combined to provide a comprehensive understanding.

In addition to self-attention, the transformer architecture employs **feedforward layers** to process the data further, adding non-linear transformations that enhance the model's understanding. To ensure stable training and preserve the quality of data as it passes through multiple layers, transformers incorporate **layer normalization and residual connections**. These features help maintain the integrity of the information, making the training process more efficient and effective. The final **output layer** generates the model's predictions or textual outputs, showcasing the model's ability to transform raw data into coherent and contextually accurate language.

### 4.3.3 Writing CVEs Based on Crash Reports Using LLMs

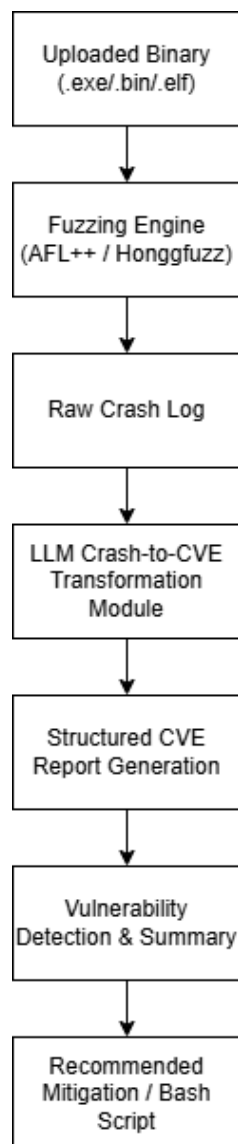
When tasked with writing a Common Vulnerabilities and Exposures (CVE) report based on a crash report, LLMs follow a structured approach. The process begins with a thorough analysis of the crash report, which typically includes details about the software's behavior, error messages, and any relevant data that led to the crash. The LLM parses this information to identify key elements, such as the nature of the error, the affected components, and the potential security implications. This initial analysis is crucial for understanding the context and severity of the vulnerability.

Once the model has a clear understanding of the crash report, it employs its attention mechanisms to examine the relationships between different pieces of information. For instance, it might focus on how a buffer overflow in a particular function could lead to a security breach or how improper input validation could expose the system to attacks. By understanding these connections, the LLM can generate a detailed description of the vulnerability. This description typically includes an outline of the flaw, the affected software versions, and the potential impact if the vulnerability is exploited. The LLM might also suggest mitigation steps or patches, drawing on its training from similar cases and established security best practices.

Finally, the generated CVE report undergoes refinement, ensuring that the language is clear, accurate, and adheres to the standard CVE format. The final product is a well-structured report that provides security professionals with the necessary information

to address the vulnerability effectively. Through this process, LLMs leverage their advanced language understanding and contextual analysis capabilities to transform raw crash data into actionable security insights, playing a crucial role in identifying and mitigating potential threats in software systems.

The diagram below illustrates the end-to-end pipeline from a binary input to the generation of CVE reports and recommendations. This visual representation summarizes the key transformation stages, starting from the user's uploaded binary file, through fuzzing and crash detection, to the generation of vulnerability reports and recommended mitigation.



**Figure 4.3.3:** End-to-End Transformation Flow

### **1. Uploaded Binary (.exe/.bin/.elf):**

The process begins when a user uploads a compiled executable or binary file to the system. These files are typically in .exe or .bin format and represent the software artifact to be tested.

### **2. Fuzzing Engine (AFL++ / Honggfuzz):**

The uploaded binary is passed to the fuzzing engine—either AFL++ or Honggfuzz—which generates and mutates test inputs to trigger abnormal behaviors such as crashes or memory violations. The fuzzer monitors for unexpected behavior during execution.

### **3. Raw Crash Log:**

If a crash or fault occurs during fuzzing, the fuzzer captures detailed crash logs, which include information such as signal type (e.g., SIGSEGV, SIGABRT), affected function names, source code line numbers, and crash types (e.g., heap-buffer-overflow).

### **4. LLM Crash-to-CVE Transformation Module:**

The raw crash log is sent to the large language model (LLM), which parses and interprets the crash data. The LLM uses predefined templates and vulnerability knowledge to map symptoms to known vulnerability patterns and formats the output in a CVE-like structure.

### **5. Structured CVE Report Generation:**

The transformed output includes a formalized CVE entry containing metadata such as vulnerability type, affected function, severity score (e.g., CVSS), and description of the impact.

### **6. Vulnerability Detection & Summary:**

The CVE data is processed and displayed to the user in a readable report format. This stage helps the user understand the nature, location, and impact of the vulnerability without needing to interpret raw crash logs.

### **7. Recommended Mitigation / Bash Script:**

Based on the type of vulnerability, the system also generates a recommended fix or patch. In some cases, this includes a Bash script or code snippet that can be executed to apply a temporary or permanent mitigation.

#### 4.3.4 Crash Log to CVE: Sample Conversion Flow

To better illustrate the functionality of the LLM in generating CVEs from crash reports, the following is a practical example showing each step of the transformation:

##### Sample Raw Crash Log

Parallel Fuzzing Report (AFL++ and Honggfuzz)

```
=====
Start Time: 2025-05-01 15:32:04
Duration: 30 minutes
Target Program: testdisk
AFL++ Output Directory: /home/einjun/AFLplusplus/output1_2025-05-01_15-32-04
Honggfuzz Output Directory: /home/einjun/Desktop/outhonggfuzz/outhonggfuzz_2025-05-01_15-32-04
=====
```

AFL++ Configuration:

```
-----
Input Directory: /home/einjun/AFLplusplus/inputdisk
Output Directory: /home/einjun/AFLplusplus/output1_2025-05-01_15-32-04
Command: afl-fuzz -Q -i /home/einjun/AFLplusplus/inputdisk -o
/home/einjun/AFLplusplus/output1_2025-05-01_15-32-04 -t 5000+ --
/home/einjun/AFLplusplus/testdisk
```

Honggfuzz Configuration:

```
-----
Input Directory: /home/einjun/AFLplusplus/inputdisk
Output Directory: /home/einjun/Desktop/outhonggfuzz/outhonggfuzz_2025-05-01_15-32-04
Command: honggfuzz -i /home/einjun/AFLplusplus/inputdisk -o
/home/einjun/Desktop/outhonggfuzz/outhonggfuzz_2025-05-01_15-32-04 -t 5 -s -- /home/einjun/AFLplusplus/testdisk
=====
```

```
Starting AFL++ at 2025-05-01 15:32:04
AFL++ process started with PID: 16508
Starting Honggfuzz at 2025-05-01 15:32:04
Honggfuzz process started with PID: 16510
2025-05-01 15:32:07 - Fuzzing for 0m 3s out of 30m
2025-05-01 15:32:10 - Fuzzing for 0m 6s out of 30m
=====
```

Combined Fuzzing Crash Report

```
=====
Crash Source: AFL++
Target Program: testdisk
Total Runtime: 9 seconds
```

## CHAPTER 4

Report Generated: 2025-05-01 15:32:14

### AFL++ Crash Information

Time of Detection: 2025-05-01 15:32:13

#### Crash Files Found:

File: id:000000,sig:11,src:000000,time:6877,execs:2836,op:havoc,rep:3

Size: 11 bytes

SHA256:

e2bab53a0d59c6b8577271e1bb52e95a77b07701bff4ce53b87dabd5e2e3ebb5

Hexdump:

00000000: 6372 6173 680a 6868 6861 73 crash.hhhas

Stack Trace:

Reading symbols from /home/einjun/AFLplusplus/testdisk...

(No debugging symbols found in /home/einjun/AFLplusplus/testdisk)

(gdb) Starting program: /home/einjun/AFLplusplus/testdisk <  
/home/einjun/AFLplusplus/output1\_2025-05-01\_15-32-

04/default/crashes/id:000000,sig:11,src:000000,time:6877,execs:2836,op:havoc,rep:3

[Thread debugging using libthread\_db enabled]

Using host libthread\_db library "/lib/x86\_64-linux-gnu/libthread\_db.so.1".

[Detaching after vfork from child process 23689]

fallocate: fallocate failed: No space left on device

[Detaching after vfork from child process 23711]

fallocate: fallocate failed: No space left on device

Memory allocation failed: Cannot allocate memory

[Detaching after vfork from child process 23723]

Program received signal SIGSEGV, Segmentation fault.

0x0000555555552ef in cause\_effect ()

(gdb) #0 0x0000555555552ef in cause\_effect ()

#1 0x000055555555389 in main ()

(gdb) A debugging session is active.

Inferior 1 [process 23509] will be killed.

Quit anyway? (y or n) [answered Y; input not from terminal]

#### AFL++ Fuzzer Statistics:

start\_time : 1746084724

last\_update : 1746084724

run\_time : 0

fuzzer\_pid : 16508

cycles\_done : 0

cycles\_wo\_finds : 0

## CHAPTER 4

```
time_wo_finds    : 0
fuzz_time        : 0
calibration_time : 0
cmplog_time      : 0
sync_time        : 0
trim_time        : 0
execs_done       : 57
execs_per_sec    : 11400.00
execs_ps_last_min : 0.00
corpus_count     : 8
corpus_favored   : 1
corpus_found     : 0
corpus_imported  : 0
corpus_variable  : 0
max_depth        : 1
cur_item         : 0
pending_favs     : 0
pending_total    : 1
stability        : 100.00%
bitmap_cvg       : 0.04%
saved_crashes    : 0
saved_hangs      : 0
last_find        : 0
last_crash       : 0
last_hang        : 0
execs_since_crash : 57
exec_timeout     : 5000
slowest_exec_ms  : 0
peak_rss_mb      : 0
cpu_affinity     : 0
edges_found      : 25
total_edges      : 65536
var_byte_count   : 0
havoc_expansion  : 0
auto_dict_entries : 0
testcache_size   : 6
testcache_count  : 1
testcache_evict  : 0
afl_banner       : /home/einjun/AFLplusplus/testdisk
afl_version      : ++4.22a
target_mode      : qemu shmem_testcase
command_line      : /home/einjun/AFLplusplus/afl-fuzz -Q -i
                  /home/einjun/AFLplusplus/inputdisk -o /home/einjun/AFLplusplus/output1_2025-05-
                  01_15-32-04 -t 5000+ -- /home/einjun/AFLplusplus/testdisk
```

```
=====
End of Report
=====
```

## CHAPTER 4

Crashes found! Check report at:  
/home/einjun/AFLplusplus/logs/parallel\_fuzzing\_2025-05-01\_15-32-04\_report.txt  
Terminating fuzzers due to crash detection  
AFL++ PID: 16508  
Honggfuzz PID: 16510

### **LLM-Generated CVE Description**

=====CVE Report=====

Program Name : testdisk

Vulnerability Name : Memory Corruption leading to Segmentation Fault

Description : The crash report indicates a segmentation fault (SIGSEGV) caused by memory corruption in the testdisk program. The issue occurs when the program attempts to access or modify memory locations that it is not supposed to, leading to a crash.

Impact : Exploiting this vulnerability could potentially allow an attacker to execute arbitrary code, cause a denial of service, or gain unauthorized access to sensitive information.

CVSS Score : 7.5

Affected Component : Cause\_effect function in the testdisk program

=====



**LLM-Suggested Mitigation**

## Recommendations:

Code Changes Required: Implement proper input validation checks and boundary controls in the cause\_effect function to prevent memory corruption issues. Use safe memory handling functions and perform thorough testing to ensure no vulnerabilities remain.

## Mitigation Link:

AddressSanitizer (ASan) can be installed to detect various memory errors like buffer overflows, use-after-free, and other memory corruption issues at runtime. By using ASan during development and testing, developers can identify and fix memory-related vulnerabilities early in the software development lifecycle.

```
=====
DISK_FLOODING_ATTACK_MITIGATION   Status:   Successfully   removed
vulnerable file.
```

```
=====
Starting Disk Flood Attack Mitigation Tool
=====
```

```
Scanning for large files in /tmp...
```

```
-----
Disk space before operation:
```

Filesystem	Size	Used	Avail	Use%	Mounted on
/dev/sda2	50G	47G	499M	99%	/

```
-----
Setting permissions and ownership for safe deletion...
```

```
-----
Removing /tmp/fuzz_test to free up space...
```

```
Success: Disk space has been freed!
```

```
-----
Disk space after operation:
```

Filesystem	Size	Used	Avail	Use%	Mounted on
/dev/sda2	50G	29G	19G	61%	/

```
=====
Disk Flood Attack Mitigation Complete
=====
```

### **Explanation of LLM Transformation Process**

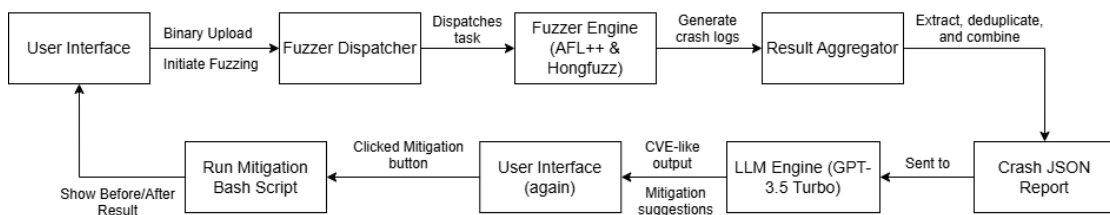
The transformation of a raw crash log into a structured CVE entry by the LLM involves a multi-stage process, outlined as follows:

- **Input Parsing:** The LLM begins by tokenizing and interpreting the raw crash log to extract critical metadata, including the type of vulnerability (e.g., heap-buffer-overflow), the function involved (parse\_data), and the exact location within the source code (main.c:88). This step provides the foundational context for further analysis.
- **Contextual Analysis:** Leveraging its pretrained knowledge on vulnerability patterns and security incidents, the LLM identifies the nature of the flaw and its implications. For example, it understands that reading beyond allocated memory constitutes a buffer overflow, which may lead to program crashes or arbitrary code execution.
- **Structured CVE Generation:** Utilizing standardized templates and domain-specific heuristics, the LLM constructs a comprehensive CVE description. This includes detailing the vulnerability, identifying affected components, assigning a severity score (e.g., CVSS), and suggesting appropriate mitigation strategies.
- **Formatting and Finalization:** Finally, the generated report is refined to adhere to CVE documentation standards. All placeholders are populated with extracted or inferred values, ensuring the output is both technically accurate and suitable for integration into vulnerability tracking systems.

#### 4.4 System Component Interaction & Data Operations

This section explains how the core components of the system interact with one another and how data flows throughout the system from user input to vulnerability identification and mitigation. These interactions are structured to maximize modularity, enable automation, and ensure seamless data handling between each component.

The figure below illustrates the complete interaction and data flow between system components, beginning from the user interface and ending with vulnerability reporting and mitigation execution. It encapsulates the flow of data through the Fuzzing Dispatcher, Fuzzing Engines (AFL++ and Honggfuzz), Result Aggregator, LLM-based CVE Generator, and Mitigation Engine, all the way to the feedback and visualization layer presented to the user.



**Figure 4.4:** System Component Data Flow and Interaction Diagram

##### 4.4.1 User Interface to Fuzzing Dispatcher

The interaction begins when a user accesses the system through the web-based user interface, developed using PHP and JavaScript. The user uploads a binary file for testing. Upon submission, the interface communicates with the Fuzzing Dispatcher, which handles task scheduling and manages the fuzzing workflow.

Once a file is submitted, the Fuzzing Dispatcher evaluates available system resources. If the number of concurrent fuzzing tasks is below the defined threshold (maximum of four), the dispatcher initiates a new fuzzing session. If the system is at capacity, new sessions are queued and users are notified via the interface.

#### 4.4.2 Fuzzing Dispatcher to Fuzzing Engines

The Fuzzing Dispatcher invokes both integrated fuzzing engines, **AFL++** and **Honggfuzz** based on user selection and task type. Binaries are fuzzed using the following modes:

- **AFL++**: Uses QEMU mode to support binary-only fuzzing without source code instrumentation.
- **Honggfuzz**: Operates via standard input to provide rapid execution with custom seed files.

Each fuzzer uses its internal mutation engine to manipulate input seed files and feed malformed or edge-case inputs into the target binary to trigger unexpected behaviors such as crashes or logic faults.

#### 4.4.3 Fuzzing Engines to Result Aggregator

As the fuzzing engines run, they generate reports, logs, and crash information. When a crash occurs:

1. **AFL++** writes crash logs and meta-info into a structured directory.
2. **Honggfuzz** appends crash data to a continuously updated crash file.

The Result Aggregator periodically monitors these outputs. When a valid crash is identified, it performs a deduplication process:

1. **AFL++** crash information is extracted based on the initialized fuzzing report structure.
2. The last 30 lines of **Honggfuzz**'s crash log are captured, as new crashes are appended at the end.
3. The two reports are merged into a unified text format and saved into a dedicated crash directory.
4. To maintain focus on the most recent analysis, the system deletes previous crash logs once the latest one is saved.

The combined crash report is then transformed into a structured JSON file containing metadata such as the crash location, input seed that caused the failure, and affected components.

### 4.4.4 JSON Report to LLM Engine

Once the JSON-formatted crash report is ready, it is submitted to a Large Language Model (LLM), specifically OpenAI's GPT-3.5 Turbo through a backend PHP-based API integration.

The LLM is prompted to analyze the structured crash data and generate:

- A **CVE-style report** detailing the nature of the vulnerability.
- A **vulnerability classification** (e.g., buffer overflow, logic bug, memory leak).
- **Mitigation strategies**, based on predefined vulnerability-to-script mappings.

The LLM output is returned to the interface, where it is parsed and prepared for user presentation.

### 4.4.5 Interface Display and Mitigation Execution

The web interface receives the LLM's output and displays it through a chatbot-like interface. The vulnerability report includes technical summaries and suggested mitigations. If a recommended mitigation corresponds to one of the system's pre-scripted Bash remediation scripts, a "**Mitigate**" button is shown.

When the user clicks this button:

- The corresponding Bash script is executed.
- A loading indicator is displayed to reflect progress.
- Once completed, the system presents a **before-and-after visualization**, such as reduced disk usage, permission changes, or sandbox activation, depending on the vulnerability addressed.

This mechanism creates a highly interactive and automated vulnerability mitigation flow for the end user.

### 4.4.6 Summary of Data Operations

To summarize, the system handles data through the following structured flow:

1. **Binary Upload & Task Initiation:** User uploads file and initiates fuzzing process.
2. **Task Scheduling & Execution:** Fuzzing Dispatcher assigns the binary to selected engines.
3. **Crash Detection:** Engines generate logs and crash reports upon encountering vulnerabilities.
4. **Data Consolidation & Structuring:** Result Aggregator merges and formats crash reports into JSON.
5. **Vulnerability Analysis:** LLM interprets crash data and generates human-readable CVE-like outputs.
6. **User Notification & Action:** Interface presents the findings and enables direct mitigation actions.

# CHAPTER 5

## System Implementation

This chapter explains how the proposed system was practically implemented, covering the complete setup from hardware selection to full system operation. It begins with the hardware and software environment used to run the application, followed by detailed configuration steps including environment variables, directory structure, and integration of bash scripts with the PHP-based frontend and LLM engine. The system operation section walks through the full user workflow from registration and binary upload to fuzzing, crash detection, CVE generation, and mitigation actions. Common implementation challenges such as inconsistent fuzzer behavior, file permission issues, and frontend integration difficulties are also discussed, providing insight into the practical obstacles and solutions encountered during development.

### 5.1 Hardware Setup

For this project focused on automated vulnerability assessments using a multi-fuzzer setup, the hardware environment centers around the use of a personal laptop. The laptop serves as the primary development and testing platform. Table 5.1 outlines the specifications of the hardware used.

**Table 5.1:** Specifications of Laptop

Description	Specifications
Model	Asus UX425EA Zenbook series
Processor	Intel® Core™ i7-1165G7 Processor 2.8 GHz (12M Cache, up to 4.7 GHz, 4 cores)
Operating System	Windows 11
Graphic	Intel Iris X <sup>e</sup> Graphics
Memory	8GB LPDDR4X on board
Storage	512GB M.2 NVMe™ PCIe® 3.0 SSD

This hardware configuration provides a reliable and efficient base for conducting extensive software-based testing, fuzzing operations, and system automation.

### 5.2 Software Setup

To develop, configure, and run fuzzing tasks for binary files, several software tools were installed and prepared within a virtualized Linux environment. The following subsections detail the software stack and corresponding setup procedures.

#### 5.2.1 Software

1. Ubuntu Operating System (Version: 24.04)
2. Fuzzing Tools
  - a. AFL++ (Version: 4.22a)
  - b. Honggfuzz (Version: 2.6)
3. Development Tools
  - a. GCC (GNU Compiler Collection)
  - b. Make
4. Version Control System
  - a. Git (Version: 1:2.43.0)
5. Automation & Scripting Tools
  - a. Bash
  - b. PHP
  - c. JavaScript
6. Additional Build Tools
  - a. CMake (Version: 3.28.3)

#### 5.2.2 Configuration and Environment Setup

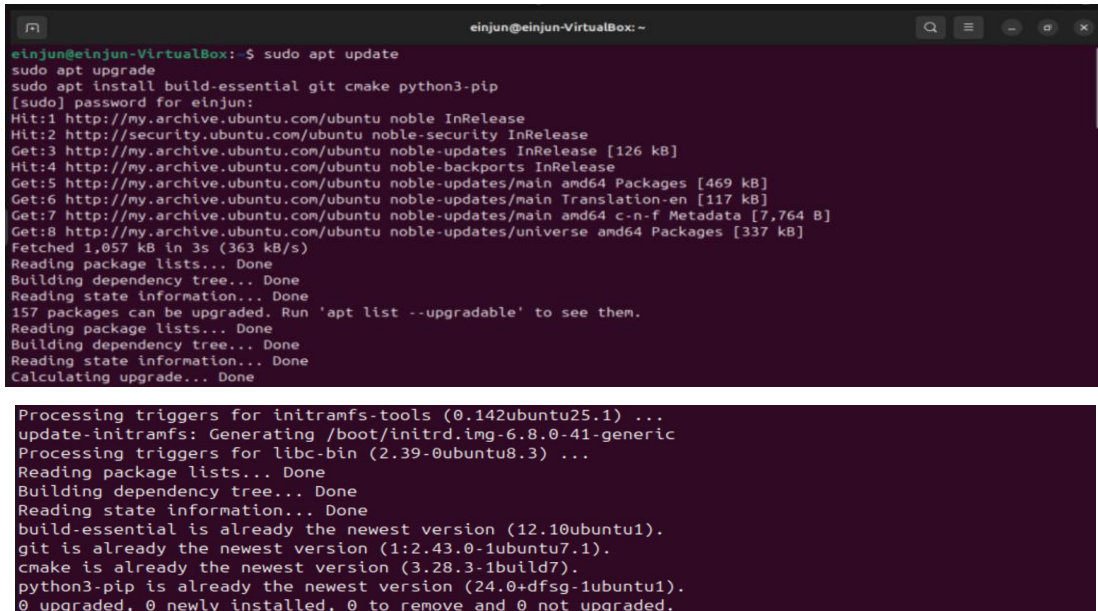
1. Ubuntu System Update and Package Installation

To begin, update the Ubuntu system to ensure all packages are up-to-date and secure. This involves using the **apt** package manager to update the list of available packages and upgrade any outdated ones. Essential build tools, such as **build-essential**, **git**, **cmake**, and **python3-pip**, are then installed to facilitate the compilation of software and management of the development environment.



Install essential build tools:

- `sudo apt update`
- `sudo apt upgrade`
- `sudo apt install build-essential git cmake python3-pip`



```

einjun@einjun-VirtualBox: ~
einjun@einjun-VirtualBox:~$ sudo apt update
sudo apt upgrade
sudo apt install build-essential git cmake python3-pip
[sudo] password for einjun:
Hit:1 http://my.archive.ubuntu.com/ubuntu noble InRelease
Hit:2 http://security.ubuntu.com/ubuntu noble-security InRelease
Get:3 http://my.archive.ubuntu.com/ubuntu noble-updates InRelease [126 kB]
Hit:4 http://my.archive.ubuntu.com/ubuntu noble-backports InRelease
Get:5 http://my.archive.ubuntu.com/ubuntu noble-updates/main amd64 Packages [469 kB]
Get:6 http://my.archive.ubuntu.com/ubuntu noble-updates/main Translation-en [117 kB]
Get:7 http://my.archive.ubuntu.com/ubuntu noble-updates/main amd64 c-n-f Metadata [7,764 B]
Get:8 http://my.archive.ubuntu.com/ubuntu noble-updates/universe amd64 Packages [337 kB]
Fetched 1,057 kB in 3s (363 kB/s)
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
157 packages can be upgraded. Run 'apt list --upgradable' to see them.
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
Calculating upgrade... Done

Processing triggers for initramfs-tools (0.142ubuntu25.1) ...
update-initramfs: Generating /boot/initrd.img-6.8.0-41-generic
Processing triggers for libc-bin (2.39-0ubuntu8.3) ...
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
build-essential is already the newest version (12.10ubuntu1).
git is already the newest version (1:2.43.0-1ubuntu7.1).
cmake is already the newest version (3.28.3-1build7).
python3-pip is already the newest version (24.0+dfsg-1ubuntu1).
0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.

```

**Figure 5.2.2.1:** Ubuntu System Update and Package Installation

## 2. Installing AFL++:

AFL++ (American Fuzzy Lop Plus Plus) is a powerful fuzzing tool required for this project. To install it, clone the AFL++ repository from GitHub and compile it from source [56]. The installation is finalized by running the **make** commands to build and install AFL++ and its associated tools. If fuzzing ELF binaries with QEMU mode is required, set the **AFL\_PATH** environment variable to the AFL++ installation path.

Clone the AFL++ repository and compile the fuzzer with QEMU support:

- `git clone https://github.com/AFLplusplus/AFLplusplus.git`
- `cd AFLplusplus`
- `make all`
- `sudo make install`
- `make distrib QEMU_BUILD=1`

```

einjun@einjun-VirtualBox:~$
einjun@einjun-VirtualBox:~$ git clone https://github.com/AFLplusplus/AFLplusplus.git

```

```

einjun@einjun-VirtualBox:~$ cd AFLplusplus
einjun@einjun-VirtualBox:~/AFLplusplus$ cd AFLplusplus
make all
sudo make install
make distrib QEMU_BUILD=1

```

```

einjun@einjun-VirtualBox:~/AFLplusplus$ ls
afl-addseeds      afl-llvm-pass.so      dictionaries          out10      out6
afl-analyze       afl-persistent-config Dockerfile             out11      out7
afl-as            afl-plot              docs                  out12      out8
afl-c++           afl-qemu-trace        dynamic_list.txt      out13      out9
afl-c++.8         afl-showmap           frida_mode            out14      output
afl-cc            afl-system-config     GNUmakefile           out15      qemu_mode
afl-cc.8          afl-tmin              GNUmakefile.gcc_plugin out16      README.md
afl-clang         afl-whatshup          GNUmakefile.llvm      out17      SanitizerCoveragePCGUARD.so
afl-clang++       afl-wine-trace        include                out18      split-compares-pass.so
afl-clang-fast    Android.bp            injection-pass.so      out19      split-switches-pass.so
afl-clang-fast++  as                    injections.dic          out20      src
afl-clang-fast+.8 benchmark              input                  out21      test
afl-clang-fast.8  Changelog.md          instrumentation         out22      test1
afl-cmin          CITATION.cff           libAFLDriver.a         out23      test.c
afl-cmin.bash     cmplog-instructions-pass.so libAFLQemuDriver.a    out24      testcases
afl-compiler-rt-64.o cmplog-routines-pass.so libcompov.so           out25      test-instr.c
afl-compiler-rt.o cmplog-switches-pass.so libgasan.so            out26      TODO.md
afl-fuzz           compare-transform-pass.so LICENSE                 out27      types.h
afl-g++            config.h               Makefile                out3       unicorn_mode
afl-gcc            CONTRIBUTING.md         nix_mode                out4       utils
afl-gotcpu         coresight_mode         out                      out5
afl-llvm-dict2file.so custom_mutators         out1

```

Figure 5.2.2.2: AFL++ Installation

### 3. Installing Honggfuzz:

Honggfuzz is another crucial fuzzing tool used in parallel with AFL++. Similar to AFL++, the installation involves cloning the Honggfuzz repository from GitHub and compiling it [57]. The **make** command is used to build and install Honggfuzz. This fuzzer provides additional mutation strategies and crash detection capabilities, complementing AFL++ and enhancing the fuzzing process.

Clone the Honggfuzz repository and compile the fuzzer:

- `git clone https://github.com/google/honggfuzz.git`
- `cd honggfuzz`
- `make sudo`
- `make install`

```

einjun@einjun-VirtualBox:~$
einjun@einjun-VirtualBox:~$ git clone https://github.com/google/honggfuzz.git

```

```

einjun@einjun-VirtualBox:~/honggfuzz$ make
sudo make install

```

```

einjun@einjun-VirtualBox:~/honggfuzz$ ls
android      display.c      hfuzz_cc          input.o          mangle.o          sanitizers.c      subproc.h
arch.h       display.h      honggfuzz         libhfcommon      netbsd            sanitizers.h      subproc.o
build       display.o      honggfuzz.c       libhfnetdriver   patches           sanitizers.o      third_party
CHANGELOG   Dockerfile    honggfuzz.h       libhfuzz         posix             screenshot-honggfuzz-1.png tools
cmdline.c   docs          HONGGFUZZ.REPORT.TXT linux             qemu_mode         socketfuzzer      socketfuzzer.c
cmdline.h   examples      includes           mac              README.md         socketfuzzer.h    socketfuzzer.o
cmdline.o   fuzz.c        input.c           Makefile         report.c           socketfuzzer.o    subproc.c
CONTRIBUTING.md fuzz.h        input.h           mangle.c         report.h           subproc.c
COPYING     fuzz.o        input.o           mangle.h         report.o

```

Figure 5.2.2.3: Honggfuzz Installation

### 5.3 Setting and Configuration

After installing all required software components, several configurations were performed to ensure seamless interaction between system modules and efficient operation of the fuzzing workflows. This includes environment variable setup, directory linking, user interface integration, and resource allocation.

#### 5.3.1 Environment Variables

To enable AFL++ to operate in QEMU mode for binary fuzzing, the following environment variable was set:

- Export AFL\_PATH=/path/to/AFLplusplus

```

einjun@einjun-VirtualBox:~/AFLplusplus$ pwd
/home/einjun/AFLplusplus
einjun@einjun-VirtualBox:~/AFLplusplus$ export AFL_PATH=/home/einjun/AFLplusplus

```

**Figure 5.3.1:** Internal Configuration and Binary Paths of AFL++

This ensures that AFL++ uses the correct internal configuration and binary paths during execution.

#### 5.3.2 Directory Structure Integration

For efficient management of fuzzing tasks and related outputs, the following directory structure was created. This organization ensures that input data, results, and logs are separated for easy access, analysis, and troubleshooting.

##### Directory Setup Overview:

The system was configured to use dedicated directories for various components of the fuzzing process. Below is an outline of the directory structure and its respective roles:

##### 1. Input Directory:

- Stores the initial seed files and corpus data used by the fuzzing engines to generate test cases.

- Seed files are necessary for AFL++ and Honggfuzz to start the fuzzing process. These are usually small inputs that test different program behaviors or common vulnerabilities.
  - mkdir ~/AFLplusplus/input

```

einjun@einjun-VirtualBox:~/AFLplusplus$ cd input
einjun@einjun-VirtualBox:~/AFLplusplus/input$ ls
seed1  seed2  seed3  seed4
einjun@einjun-VirtualBox:~/AFLplusplus/input$ cat seed1
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
einjun@einjun-VirtualBox:~/AFLplusplus/input$ cat seed2
1234567890
einjun@einjun-VirtualBox:~/AFLplusplus/input$ cat seed3
test
einjun@einjun-VirtualBox:~/AFLplusplus/input$ cat seed4
buffer_overflow_test
einjun@einjun-VirtualBox:~/AFLplusplus/input$

```

Figure 5.3.2.1: Target Corpus Data Directory

```

einjun@einjun-VirtualBox:~/AFLplusplus$ ls input*
input:
seed1  seed2  seed3  seed4  seed5.txt

input1:
seed5.txt

inputabort:
seed5.txt

inputalrm:
seed1.txt

inputdisk:
'seed1 (Copy).txt'  'seed2 (Copy).txt'  'seed3 (Copy).txt'  'seed5 (Copy).txt'
seed1.txt          seed2.txt          seed3.txt          seed5.txt

inputdivision:
seed1.txt

inputkeyword:
seed1.txt  seed2.txt  seed3.txt  seed5.txt

inputsigill:
seed5.txt
einjun@einjun-VirtualBox:~/AFLplusplus$

```

Figure 5.3.2.2: Test Cases Corpus Data Directory

## 2. Output Directory:

- Stores fuzzing results, including any crashes, error logs, or coverage data generated by AFL++ and Honggfuzz.
- Crash logs are organized here and later processed by the **Result Aggregator** for deduplication and reporting.
  - `mkdir ~/AFLplusplus/output`

```

einjun@einjun-VirtualBox:~/AFLplusplus$ mkdir output
einjun@einjun-VirtualBox:~/AFLplusplus$ cd output
einjun@einjun-VirtualBox:~/AFLplusplus/output$

```

**Figure 5.3.2.3:** Output Crash Directory

## 3. Binary Directory:

- Holds the target binaries (e.g., downloaded from sources like PicoCTF) and test cases that are to be fuzzed.
- This directory contains all the compiled programs that are analyzed for vulnerabilities during fuzzing.
  - `mkdir ~/Downloads/ (37 target binaries from picoCTF website [58])`

```

einjun@einjun-VirtualBox:~/AFLplusplus$ ls target*
target1  target13  target17  target20  target24  target28  target31  target35  target5  target9
target10 target14  target18  target21  target25  target29  target32  target36  target6
target11 target15  target19  target22  target26  target3  target33  target37  target7
target12 target16  target2  target23  target27  target30  target34  target4  target8

```

**Figure 5.3.2.4:** Target Application Directory

```

einjun@einjun-VirtualBox:~/AFLplusplus$ ls test*
test1.c  test4.c  test7.c  testabort  testdisk  test-instr.c  testsigill
test2.c  test5.c  test8.c  testalarm  testdivision  testkeyword  testsleep
test3.c  test6.c  test9.c  testbuffer  testfile  testrun

```

**Figure 5.3.2.5:** Test Cases Application Directory

#### 4. Crash Reports Directory:

- Stores the combined crash logs generated by the fuzzers (AFL++ and Honggfuzz). These logs contain detailed information about crashes or program behavior that could indicate vulnerabilities.
  - Crash logs by AFL++

```

einjun@einjun-VirtualBox:~$ cat /home/einjun/AFLplusplus/output1_2025-04-26_00-14-45/default/fuzzer_stats
start_time      : 1745597685
last_update     : 1745597685
run_time        : 0
fuzzer_pid      : 79911
cycles_done     : 0
cycles_wo_finds : 0
time_wo_finds   : 0
fuzz_time       : 0
calibration_time : 0
cmplog_time     : 0
sync_time       : 0
trim_time       : 0
execs_done      : 8
execs_per_sec   : 4000.00
execs_ps_last_min : 0.00
corpus_count    : 1
corpus_favored  : 1
corpus_found    : 0
corpus_imported : 0
corpus_variable : 0
max_depth       : 1
cur_item        : 0
pending_favs    : 1
pending_total   : 1
stability        : 100.00%
bitmap_cvg      : 0.05%
saved_crashes   : 0
saved_hangs     : 0
last_find       : 0
last_crash      : 0
last_hang       : 0
execs_since_crash : 8
exec_timeout    : 5000
slowest_exec_ms : 0
peak_rss_mb     : 0
cpu_affinity    : 0
edges_found     : 30
total_edges     : 65536
var_byte_count  : 0
havoc_expansion : 0
auto_dict_entries : 0
testcache_size  : 5
testcache_count : 1
testcache_evict : 0
afl_banner      : /home/einjun/AFLplusplus/testsleep
afl_version     : ++4.22a
target_mode     : qemu shmem_testcase
command_line    : /home/einjun/AFLplusplus/afl-fuzz -Q -i /home/einjun/AFLplusplus/input1 -o /home/einjun/AFLplusplus/output1_2025-04-26_00-14-45 -t 5000+ -- /home/einjun/AFLplusplus/testsleep
einjun@einjun-VirtualBox:~$

```

**Figure 5.3.2.6:** AFL++ Crash Logs



- Crash logs by Honggfuzz

```

einjun@einjun-VirtualBox:~$ cat /home/einjun/Desktop/outhonggfuzz/HONGGFUZZ.REPORT.TXT
=====
TIME: 2025-04-26.00:14:47
=====
FUZZER ARGS:
mutationsPerRun : 5
externalCmd      : NULL
fuzzStdin       : TRUE
timeout         : 5 (sec)
ignoreAddr      : (nil)
ASLlimit        : 0 (MiB)
RSSlimit        : 0 (MiB)
DATAlimit       : 0 (MiB)
wordlistFile     : NULL
dynFileMethod   :
fuzzTarget      : /home/einjun/AFLplusplus/testsleep
CRASH:
DESCRIPTION:
ORIG_FNAME: 00000000000000000000000000000000.00000000.honggfuzz.cov
FUZZ_FNAME: /home/einjun/Desktop/outhonggfuzz/SIGSEGV.PC.555555555318.STACK.cbb542e7a.CODE.1.ADDR.0.INSTR.mov____%eax
, (%rdx).fuzz
PID: 79928
SIGNAL: SIGSEGV (11)
PC: 0x555555555318
FAULT ADDRESS: 0x0
INSTRUCTION: mov____%eax, (%rdx)
STACK HASH: 000000cbb542e7a
STACK:
<0x00007ffff7c2a1ca> [func:UNKNOWN file: line:0 module:/usr/lib/x86_64-linux-gnu/libc.so.6]
<0x00007ffff7c2a28b> [func:UNKNOWN file: line:0 module:/usr/lib/x86_64-linux-gnu/libc.so.6]
<0x0000555555555165> [func:UNKNOWN file: line:0 module:/home/einjun/AFLplusplus/testsleep]
=====
einjun@einjun-VirtualBox:~$

```

Figure 5.3.2.7: Honggfuzz Crash Logs

- A specific script combines crash logs from both fuzzers and formats them into structured JSON files for further analysis by the **LLM Engine**.

- Combined crash logs by AFL++ and Honggfuzz

```

einjun@einjun-VirtualBox:~$ cat /home/einjun/AFLplusplus/logs/parallel_fuzzing_2025-04-26_00-14-45_report.txt
Parallel Fuzzing Report (AFL++ and Honggfuzz)
=====
Start Time: 2025-04-26 00:14:45
Duration: 30 minutes
Target Program: testsleep
AFL++ Output Directory: /home/einjun/AFLplusplus/output1_2025-04-26_00-14-45
Honggfuzz Output Directory: /home/einjun/Desktop/outhonggfuzz/outhonggfuzz_2025-04-26_00-14-45
=====
AFL++ Configuration:
-----
Input Directory: /home/einjun/AFLplusplus/input1
Output Directory: /home/einjun/AFLplusplus/output1_2025-04-26_00-14-45
Command: afl-fuzz -Q -i /home/einjun/AFLplusplus/input1 -o /home/einjun/AFLplusplus/output1_2025-04-26_00-14-45 -t 5000+
-- /home/einjun/AFLplusplus/testsleep

Honggfuzz Configuration:
-----
Input Directory: /home/einjun/AFLplusplus/input1
Output Directory: /home/einjun/Desktop/outhonggfuzz/outhonggfuzz_2025-04-26_00-14-45
Command: honggfuzz -i /home/einjun/AFLplusplus/input1 -o /home/einjun/Desktop/outhonggfuzz/outhonggfuzz_2025-04-26_00-14-45 -t 5 -s -- /home/einjun/AFLplusplus/testsleep
=====
Starting AFL++ at 2025-04-26 00:14:45
AFL++ process started with PID: 79911
Starting Honggfuzz at 2025-04-26 00:14:45
Honggfuzz process started with PID: 79913
2025-04-26 00:14:48 - Fuzzing for 0m 3s out of 30m
=====
Combined Fuzzing Crash Report
=====

```

Figure 5.3.2.8: Combined Crash Report from Both Fuzzers (Section 1)

```

=====
Combined Fuzzing Crash Report
=====
Crash Source: Both Fuzzers
Target Program: testsleep
Total Runtime: 6 seconds
Report Generated: 2025-04-26 00:14:54
=====

AFL++ Crash Information
=====
Time of Detection: 2025-04-26 00:14:51

Crash Files Found:
-----
File: id:000000,sig:11,src:000000,time:2008,execs:10,op:quick,pos:0
Size: 4 bytes
SHA256: 2f09e789907b5b31446910351837235b88a6e6bcf0336239fb76cadae12afe1
Hexdump:
00000000: 6461 6665                                     daf6
Stack Trace:
Reading symbols from /home/einjun/AFLplusplus/testsleep...
(gdb) Starting program: /home/einjun/AFLplusplus/testsleep < /home/einjun/AFLplusplus/output1_2025-04-26_00-14-45/default/crashes/id:000000,sig:11,src:000000,time:2008,execs:10,op:quick,pos:0
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".

Program received signal SIGSEGV, Segmentation fault.
0x0000555555555318 in main () at test1.c:26
26      *ptr = rand(); // This triggers a crash
(gdb) #0  0x0000555555555318 in main () at test1.c:26
(gdb) A debugging session is active.

AFL++ Fuzzer Statistics:
-----
start_time      : 1745597685
last_update     : 1745597685
run_time        : 0
fuzzer_pid      : 79911
cycles_done     : 0
cycles_wo_finds : 0
time_wo_finds   : 0
fuzz_time       : 0
calibration_time : 0
cmplog_time     : 0
sync_time       : 0
trim_time       : 0
execs_done      : 8
execs_per_sec   : 4000.00
execs_ps_last_min : 0.00
corpus_count    : 1
corpus_favored  : 1
corpus_found    : 0
corpus_imported : 0
corpus_variable : 0
max_depth       : 1
cur_item        : 0
pending_favs    : 1
pending_total   : 1
stability        : 100.00%
bitmap_cvg      : 0.05%
saved_crashes   : 0
saved_hangs     : 0
last_find       : 0
last_crash      : 0

last_hang       : 0
execs_since_crash : 8
exec_timeout    : 5000
slowest_exec_ms : 0
peak_rss_mb     : 0
cpu_affinity     : 0
edges_found     : 30
total_edges     : 65536
var_byte_count  : 0
havoc_expansion : 0
auto_dict_entries : 0
testcache_size  : 5
testcache_count : 1
testcache_evict : 0
afl_banner      : /home/einjun/AFLplusplus/testsleep
afl_version     : ++4.22a
target_mode     : qemu shmem_testcase
command_line    : /home/einjun/AFLplusplus/afl-fuzz -Q -i /home/einjun/AFLplusplus/input1 -o /home/einjun/AFLplusplus/output1_2025-04-26_00-14-45 -t 5000+ -- /home/einjun/AFLplusplus/testsleep

```

**Figure 5.3.2.9:** Combined Crash Report from Both Fuzzers (Section 2)



## CHAPTER 5

```
=====
Honggfuzz Crash Information
=====
Time of Detection: 2025-04-26 00:14:53

File: HONGGFUZZ.REPORT.TXT
Size: 1188 bytes
SHA256: 5ba35b7d8b27800cd37dc8638d9444e4137bccbffe5e719b076fbf4c65ed9bb9
Report Content:
=====
TIME: 2025-04-26.00:14:47
=====
FUZZER ARGS:
mutationsPerRun : 5
externalCmd      : NULL
fuzzStdin       : TRUE
timeout         : 5 (sec)
ignoreAddr      : (nil)
ASLlimit        : 0 (MiB)
RSSlimit        : 0 (MiB)
DATAlimit       : 0 (MiB)
wordlistFile     : NULL
dynFileMethod   :
fuzzTarget      : /home/einjun/AFLplusplus/testsleep

CRASH:
DESCRIPTION:
ORIG_FNAME: 00000000000000000000000000000000.00000000.honggfuzz.cov
FUZZ_FNAME: /home/einjun/Desktop/outputhonggfuzz/SIGSEGV.PC.555555555318.STACK.cbb542e7a.CODE.1.ADDR.0.INSTR.mov____%eax
,(%rdx).fuzz
PID: 79928
SIGNAL: SIGSEGV (11)
PC: 0x555555555318

=====
End of Report
=====
Crashes found! Check report at: /home/einjun/AFLplusplus/logs/parallel_fuzzing_2025-04-26_00-14-45_report.txt
Terminating fuzzers due to crash detection
AFL++ PID: 79911
Honggfuzz PID: 79913
einjun@einjun-VirtualBox: $
```

**Figure 5.3.2.10:** Combined Crash Report from Both Fuzzers (Section 3)

### 5.3.3 Bash Script Configuration

To streamline the operation of the multi-fuzzer system and reduce manual intervention, a set of Bash scripts was developed to automate key processes in the pipeline. These scripts serve as backend logic to connect fuzzing outputs with vulnerability analysis and mitigation workflows.

#### a. Fuzzing Execution Script

- This script is responsible for launching AFL++ or Honggfuzz based on the user's selection.
- It accepts parameters such as the target binary path, input seed folder, and output directory.
- It ensures that no more than four fuzzers run concurrently to optimize system performance.

```
# Start AFL++
echo "Starting AFL++ at $(date '+%Y-%m-%d %H:%M:%S')" >> "$LOG_FILE"
/home/einjun/AFLplusplus/afl-fuzz -Q -i "$INPUT_DIR" -o "$AFL_OUTPUT_DIR" -t 5000+ -- "$TARGET" &
AFL_PID=$!
echo "AFL++ process started with PID: $AFL_PID" >> "$LOG_FILE"

# Start Honggfuzz
echo "Starting Honggfuzz at $(date '+%Y-%m-%d %H:%M:%S')" >> "$LOG_FILE"
cd /home/einjun/Desktop/outputhonggfuzz
honggfuzz -i "$INPUT_DIR" -o "$HONGGFUZZ_OUTPUT_DIR" -t 5 -s -- "$TARGET" &
HONGGFUZZ_PID=$!
echo "Honggfuzz process started with PID: $HONGGFUZZ_PID" >> "$LOG_FILE"
```

**Figure 5.3.3.1:** Parallel Fuzzing Execution Script

#### b. Crash Report Aggregator

This script handles the collection and merging of crash reports from both fuzzers:

- AFL++: Parses fuzzing queue and crash summary.

```

check_crashes() {
    # Check AFL++ crashes
    if [ -d "$AFL_OUTPUT_DIR/default/crashes" ] && [ "$(ls -A $AFL_OUTPUT_DIR/default/crashes)" ]; then
        crash_detected=true
        crash_source="AFL++"

        crash_details+="\n\n"
        crash_details+="AFL++ Crash Information\n"
        crash_details+="\n\n"
        crash_details+="Time of Detection: $(date '+%Y-%m-%d %H:%M:%S')\n\n"

        # Add AFL++ crash files details
        crash_details+="Crash Files Found:\n"
        crash_details+="-----\n"
        for crash_file in "$AFL_OUTPUT_DIR/default/crashes"/*; do
            if [ -f "$crash_file" ] && [ "$(basename "$crash_file")" != "README.txt" ]; then
                crash_details+="File: $(basename "$crash_file")\n"
                crash_details+="Size: $(ls -l "$crash_file" | awk '{print $5}') bytes\n"
                crash_details+="SHA256: $(sha256sum "$crash_file" | cut -d' ' -f1)\n"
                crash_details+="Hexdump:\n"
                crash_details+="$(xxd "$crash_file")\n"

                # Try to get stack trace if available
                if command -v gdb && /dev/null; then
                    crash_details+="Stack Trace:\n"
                    crash_details+="$(echo -e "run < $crash_file\nbt\nquit" | gdb -q "$TARGET" 2>&1)\n"
                fi
                crash_details+="-----\n"
            fi
        done
    fi

    # Add AFL++ statistics
    if [ -f "$AFL_OUTPUT_DIR/default/fuzzer_stats" ]; then
        crash_details+="\nAFL++ Fuzzer Statistics:\n"
        crash_details+="-----\n"
        crash_details+="$(cat "$AFL_OUTPUT_DIR/default/fuzzer_stats")\n"
    fi
}

```

Figure 5.3.3.2: AFL++ Information Extracting Script

- Honggfuzz: Extracts information from the latest crash report.

```

check_crashes() {
    # Check Honggfuzz crashes - Improved detection
    if [ -d "$HONGGFUZZ_OUTPUT_DIR" ]; then
        # Look for crash files with specific patterns
        local honggfuzz_crashes=(
            "$HONGGFUZZ_CRASH_DIR"/HONGGFUZZ.REPORT.*
            "$HONGGFUZZ_CRASH_DIR"/SIGABRT.*
            "$HONGGFUZZ_CRASH_DIR"/SIGSEGV.*
            "$HONGGFUZZ_CRASH_DIR"/*.fuzz
        )

        # Check if any crash files exist
        for pattern in "${honggfuzz_crashes[@]"; do
            if ls $pattern 1> /dev/null 2>&1; then
                crash_detected=true
                [ -z "$crash_source" ] && crash_source="Honggfuzz" || crash_source="Both Fuzzers"

                crash_details+="\n\n"
                crash_details+="Honggfuzz Crash Information\n"
                crash_details+="\n\n"
                crash_details+="Time of Detection: $(date '+%Y-%m-%d %H:%M:%S')\n\n"

                # Process each crash file
                for crash_file in $pattern; do
                    if [ -f "$crash_file" ]; then
                        crash_details+="File: $(basename "$crash_file")\n"
                        crash_details+="Size: $(ls -l "$crash_file" | awk '{print $5}') bytes\n"
                        crash_details+="SHA256: $(sha256sum "$crash_file" | cut -d' ' -f1)\n"

                        # If it's a report file, include its content
                        if [[ "$crash_file" == *.REPORT* ]]; then
                            crash_details+="Report Content:\n"
                            crash_details+="$(tail -n 30 "$crash_file")\n"
                        else
                            # For crash input files, show hexdump
                            crash_details+="Hexdump:\n"
                            crash_details+="$(xxd "$crash_file")\n"

                            # Try to reproduce the crash and get stack trace
                            crash_details+="Crash Reproduction:\n"
                            crash_details+="$(echo -e "run < \"$crash_file\" 2>&1)\n"

                            # Get stack trace using GDB
                            if command -v gdb && /dev/null; then
                                crash_details+="Stack Trace:\n"
                                crash_details+="$(echo -e "run < $crash_file\nbt\nquit" | gdb -q "$TARGET" 2>&1)\n"
                            fi
                        fi
                    fi
                done
            fi
        done
    fi
}

```

```

# Add Honggfuzz statistics if available
if [ -f "$HONGGFUZZ_OUTPUT_DIR/HONGGFUZZ.STATS.txt" ]; then
    crash_details+="\nHonggfuzz Statistics:\n"
    crash_details+="-----\n"
    crash_details+=$(cat "$HONGGFUZZ_OUTPUT_DIR/HONGGFUZZ.STATS.txt")\n"
fi
break
fi
done
fi

```

**Figure 5.3.3.3:** Honggfuzz Information Extracting Script

- Once fuzzing completes or a crash is detected, this script extracts crash logs from both AFL++ and Honggfuzz output directories.
- The logs are filtered and deduplicated to avoid repetition and combined into a single text file.
- This file is then converted to JSON format to be processed by the LLM.

```

{
    echo "=====
    echo "Combined Fuzzing Crash Report"
    echo "=====
    echo "Crash Source: $crash_source"
    echo "Target Program: $TARGET_PROGRAM"
    echo "Total Runtime: ${ELAPSED_SECONDS} seconds"
    echo "Report Generated: $(date '+%Y-%m-%d %H:%M:%S')"
    echo
    echo -e "$crash_details"
    echo "=====
    echo "End of Report"
    echo "=====
    return 0
} >> "$LOG_FILE"

```

**Figure 5.3.3.4:** Full Crash Information Compiling Script

### c. Mitigation Trigger Script

- After receiving the CVE-like report and mitigation advice from the LLM, the system presents a "Mitigate" button in the UI.
- When the button is clicked, this Bash script executes predefined commands to apply fixes, such as modifying source code, changing file permissions, or applying input validation logic.
- It then compares and logs the "before" and "after" states to show the impact of the fix.

```
#!/bin/bash
set -e
echo "=====
echo "Starting Disk Flood Attack Mitigation Tool"
echo "=====

echo "Scanning for large files in /tmp..."
FILE="/tmp/fuzz_test"

echo "-----"
echo "Disk space before operation:"
df -h /tmp
echo "-----"

if [ -f "$FILE" ]; then
    echo "Setting permissions and ownership for safe deletion..."

    if ! sudo chmod 666 "$FILE" 2>/dev/null; then
        echo "Sudo failed, trying direct command..."
        chmod 666 "$FILE" || echo "Failed to set permissions"
    fi

    if ! sudo chown daemon:daemon "$FILE" 2>/dev/null; then
        echo "Sudo ownership change failed, continuing anyway..."
    fi

    echo "-----"
    echo "Removing $FILE to free up space..."
    if rm -f "$FILE" 2>/dev/null; then
        echo "Success: Disk space has been freed!"
    else
        echo "Warning: Failed to remove file. Trying alternative method..."
        echo "<?php unlink('$FILE'); echo 'PHP unlink result: ' . (file_exists('$FILE') ? 'Failed' : 'Success'); ?>" > /tmp/
        remove_file.php
        php /tmp/remove_file.php
        rm -f /tmp/remove_file.php
    fi
else
    echo "Status: No large files found."
fi

echo "-----"
echo "Disk space after operation:"
df -h /tmp
echo "-----"

echo "=====
echo "Disk Flood Attack Mitigation Complete"
echo "=====
```

Figure 5.3.3.5: Mitigation Trigger Script

### 5.3.4 PHP & LLM Integration

The PHP backend plays a crucial role in integrating various components of the multi-fuzzer system, ensuring that fuzzing tasks are automated and seamlessly connected to the vulnerability analysis engine powered by the LLM. The following describes how PHP was configured to handle these interactions.

#### a. File Upload and Fuzzer Trigger

- **File Upload Handling:** The PHP backend is responsible for receiving user-submitted files (target binaries) via the user interface. This is managed using PHP's file upload mechanism, which ensures that files are transferred securely to the server.
- **Fuzzer Trigger:** Upon successful file upload, the PHP backend communicates with the fuzzing dispatcher (using shell commands) to trigger both fuzzers. The file is then placed in the appropriate directory for processing.

```
// Handle file upload
if (isset($_FILES['security_file'])) {
    $target_dir = "/home/einjun/AFLplusplus/";
    $target_file = $target_dir . basename(path: $_FILES["security_file"]["name"]);
    $uploadOk = 1;

    if ($_FILES["security_file"]["size"] > 500000) {
        echo "Sorry, your file is too large.";
        $uploadOk = 0;
    }

    if ($uploadOk == 1) {
        if (move_uploaded_file(from: $_FILES["security_file"]["tmp_name"], to: $target_file)) {
            chmod(filename: $target_file, permissions: 0777);
            $filename = pathinfo(path: $_FILES["security_file"]["name"], flags: PATHINFO_FILENAME);
            header(header: "Location: index1.php?command=RUN_PARALLEL 30 " . urlencode(string: $filename));
            exit;
        } else {
            echo "Sorry, there was an error uploading your file.";
        }
    }
}
```

**Figure 5.3.4.1:** File Upload Handling Code

#### b. Fuzzing Result Capture and Crash Report Processing

- **Result Capture:** Once the fuzzing process completes, the PHP backend monitors the output directories of both AFL++ and Honggfuzz for crash reports. These logs, which are captured by the backend, contain the details of the crashes, including error messages and stack traces.

- **Crash Report Processing:** The raw crash logs are processed to extract valuable information (shown in section 5.3.3), such as crash type, vulnerability potential, and input seeds that triggered the crashes. This information is then prepared for further analysis by converting the logs into a structured format (JSON).

### c. Conversion to JSON Format

- **Data Structuring:** To prepare the crash data for the LLM, the processed logs are converted into JSON format. This structured format allows the system to send detailed, consistent input to the LLM for vulnerability analysis.
- **Standardized Format:** This includes critical data such as the vulnerability type, location, severity, and any specific trigger conditions found in the fuzzing session. A consistent structure ensures smooth interaction between PHP and the LLM API.

```
$ch = curl_init(url: $url);
curl_setopt(handle: $ch, option: CURLOPT_RETURNTRANSFER, value: true);
curl_setopt(handle: $ch, option: CURLOPT_POST, value: true);
curl_setopt(handle: $ch, option: CURLOPT_POSTFIELDS, value: json_encode(value: $data));
curl_setopt(handle: $ch, option: CURLOPT_HTTPHEADER, value: $headers);

// Fix SSL verification issues
curl_setopt(handle: $ch, option: CURLOPT_SSL_VERIFYPEER, value: false);
curl_setopt(handle: $ch, option: CURLOPT_SSL_VERIFYHOST, value: 0);

$response = curl_exec(handle: $ch);

if (curl_errno(handle: $ch)) {
    error_log(message: 'Curl error: ' . curl_error(handle: $ch));
    return 'Error: ' . curl_error(handle: $ch);
}

curl_close(handle: $ch);
```

Figure 5.3.4.2: JSON Format Conversion Code

### d. Sending Structured Data to OpenAI GPT-3.5 Turbo API

- **LLM Integration:** The PHP backend integrates with the OpenAI GPT-3.5 Turbo API, which is tasked with analyzing the structured crash data. The backend sends the JSON payload containing fuzzing results, with a prompt asking GPT-3.5 to generate a CVE-style report and possible mitigations.
- **API Call Handling:** To maintain security, API keys used to authenticate the PHP backend with OpenAI are stored securely in a .env or config file, ensuring that sensitive credentials are not exposed in the code. The backend executes the



API request and waits for the response, which includes CVE-like reports and mitigation advice.

```
if (!isset($_POST['message'])) {
    $user_message = $conn->real_escape_string(string: $_POST['message']);
    $timestamp = date(format: 'Y-m-d H:i:s');

    if (!isset($_SESSION['crash_analyzed'])) {
        $report_content = getLatestFuzzingReport();
        if ($report_content) {
            $user_message = "Please analyze this fuzzing crash report for potential vulnerabilities:\n\n" . $report_content;
            $_SESSION['crash_analyzed'] = true;
        }
    }

    $response = callOpenAI(message: $user_message);

    header(header: 'Content-Type: application/json');
    echo json_encode(value: ['response' => $response]);
    exit;
}
```

**Figure 5.3.4.3:** Request sent to ChatGPT API (OpenAI API) Code

#### e. Displaying CVE-style Reports and Mitigation Options

- **UI Display:** Upon receiving the analysis from GPT-3.5, the PHP backend processes the response and displays it in the user interface. The report includes detailed CVE-style information, such as:
  - Vulnerability description
  - Affected components
  - Risk severity
  - Possible attack vectors
- **Interactive Mitigation Options:** The user interface also displays interactive options, such as a “Mitigate” button, for each vulnerability. When clicked, this triggers predefined mitigation scripts (written in Bash) to resolve the detected issue. PHP ensures smooth interaction between the user interface and backend logic to execute these scripts.

#### f. API Key Security

- **Secure Storage:** To ensure the security of the application and prevent unauthorized access to the OpenAI API, API keys are stored in a .env file or a configuration file outside the source code. This file is secured and not pushed to version control systems, thereby safeguarding the keys from exposure.

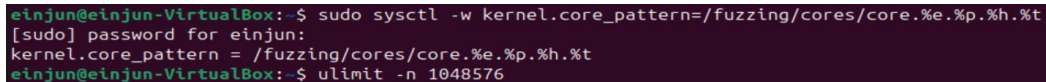


### 5.3.5 System Resource Tuning

Optimizing system resources is essential for running multiple fuzzers efficiently. Adjust kernel settings to handle core dumps correctly by modifying the **kernel.core\_pattern** using the **sysctl** command. Additionally, increase the file descriptor limit with the **ulimit** command to support extensive fuzzing sessions, which may require handling many simultaneous files. This step ensures the system can manage the high resource demands of fuzzing without encountering bottlenecks or crashes.

Adjust system configurations to allocate sufficient CPU cores and memory for parallel fuzzing with AFL++ and Honggfuzz:

- Core dump settings were adjusted for better crash traceability using:
  - `sudo sysctl -w kernel.core_pattern=/fuzzing/cores/core.%e.%p.%h.%t`
- Ulimit was increased to allow a higher number of open file descriptors.
  - `ulimit -n 1048576`



```
einjun@einjun-VirtualBox:~$ sudo sysctl -w kernel.core_pattern=/fuzzing/cores/core.%e.%p.%h.%t
[sudo] password for einjun:
kernel.core_pattern = /fuzzing/cores/core.%e.%p.%h.%t
einjun@einjun-VirtualBox:~$ ulimit -n 1048576
```

**Figure 5.3.5:** System Resources Configuration

This helped maintain system performance under load from up to four concurrent fuzzing tasks.

## 5.4 System Operation

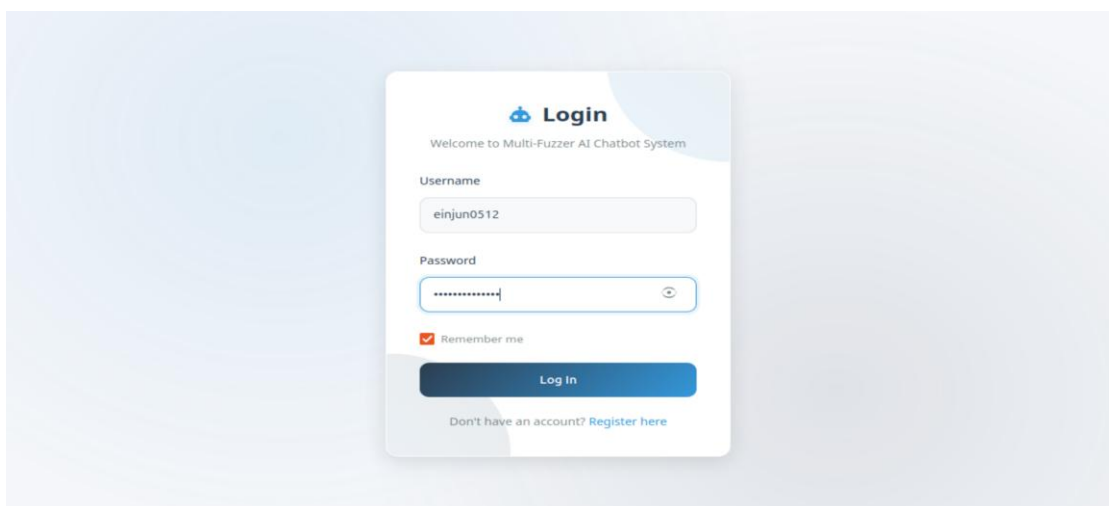
This section provides a detailed walkthrough of how the system operates, from user login and registration to the fuzzing process, crash detection, CVE report generation, and mitigation actions. The following steps outline the operation, accompanied by relevant screenshots for each phase.

### 5.4.1 User Login and Registration

Before users can interact with the system, they must log in or register for an account. The registration process includes basic user details and 2FA (Two-Factor Authentication) for enhanced security. The login page also ensures that only authorized users can access the fuzzing features.

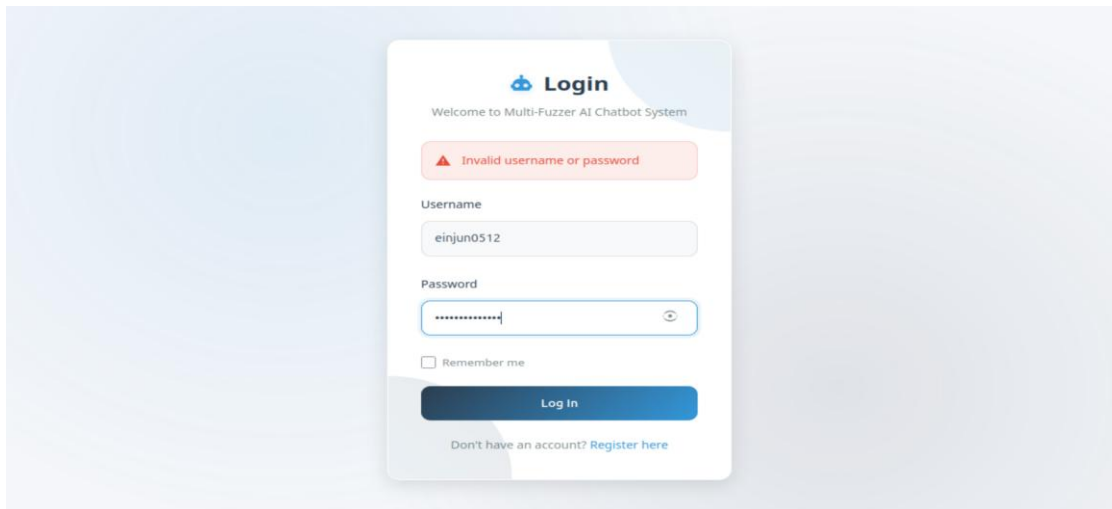
- **Login Process:**

The login page prompts users for their credentials (username and password). Once entered, the system verifies the credentials against the database. Upon successful verification, users are granted access to the fuzzing platform.



**Figure 5.4.1.1:** Login Process using Username and Password

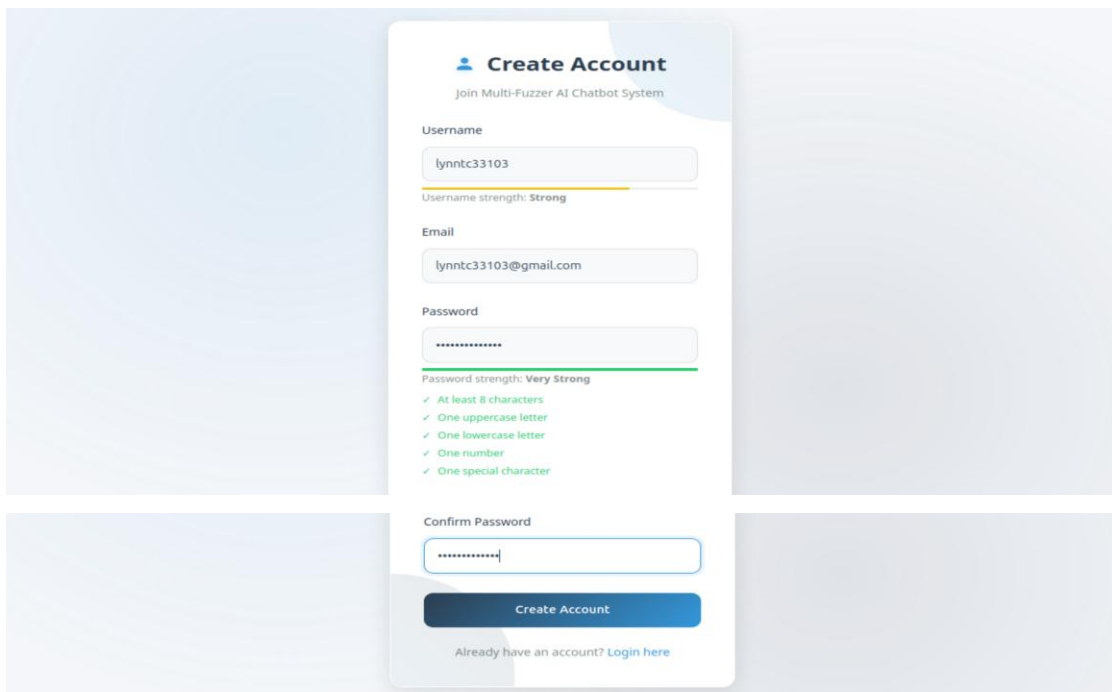
Situation where the username or password does not match:



**Figure 5.4.1.2:** Login Process Unsuccessful

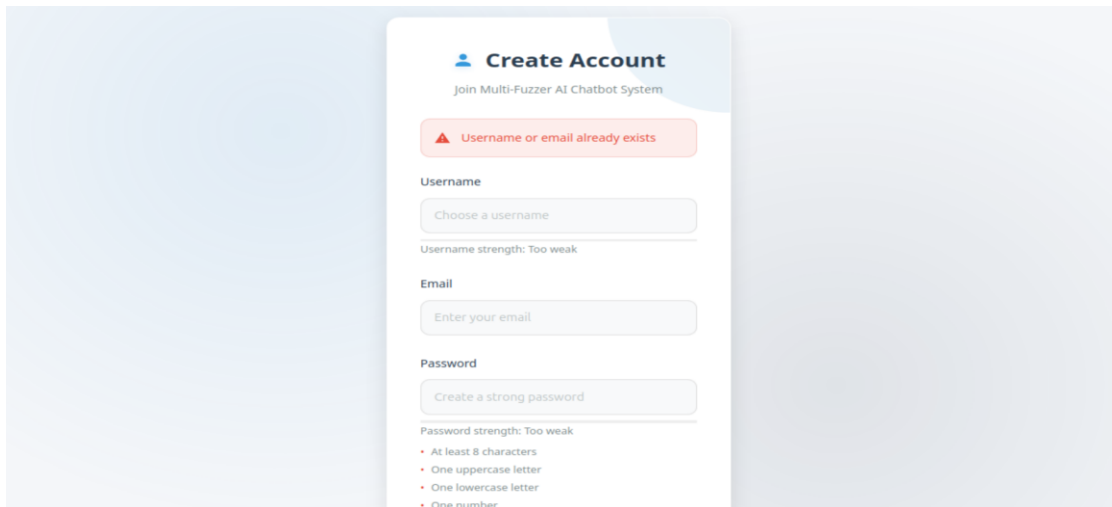
- **Registration Process:**

New users can register by providing their information (username, email, password), and once registered, they will be redirected to the login page.



**Figure 5.4.1.3:** User Registration Process

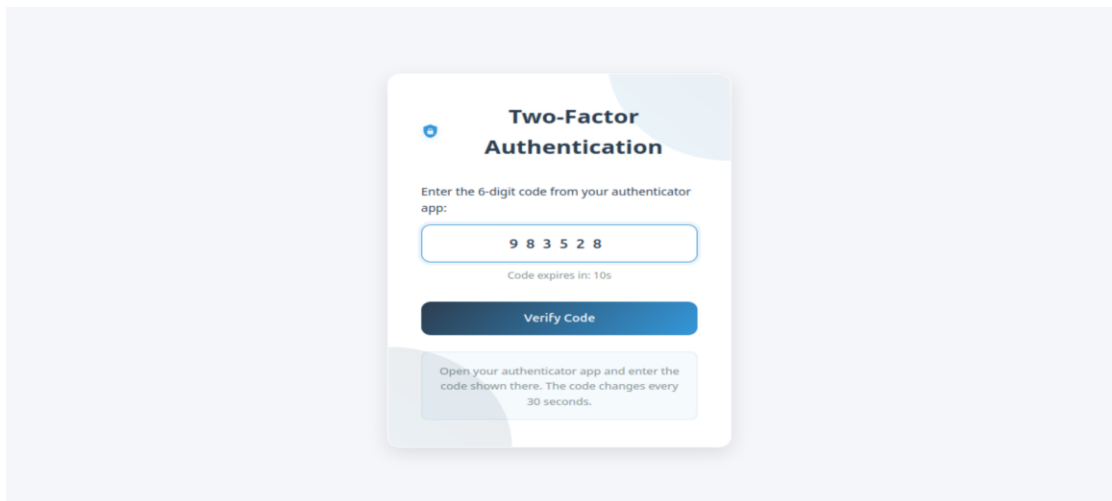
Situation where the username or email already exists in the database:

A screenshot of a 'Create Account' form for the 'Multi-Fuzzer AI Chatbot System'. The form has a title 'Create Account' with a user icon and a subtitle 'Join Multi-Fuzzer AI Chatbot System'. A red error message at the top states 'Username or email already exists'. Below this are three input fields: 'Username' with placeholder 'Choose a username', 'Email' with placeholder 'Enter your email', and 'Password' with placeholder 'Create a strong password'. The 'Username' field has a feedback message 'Username strength: Too weak'. The 'Password' field has a feedback message 'Password strength: Too weak' and a list of requirements: 'At least 8 characters', 'One uppercase letter', 'One lowercase letter', and 'One number'.

**Figure 5.4.1.4:** User Registration Process with Duplicated Registration

- **2FA Implementation:**

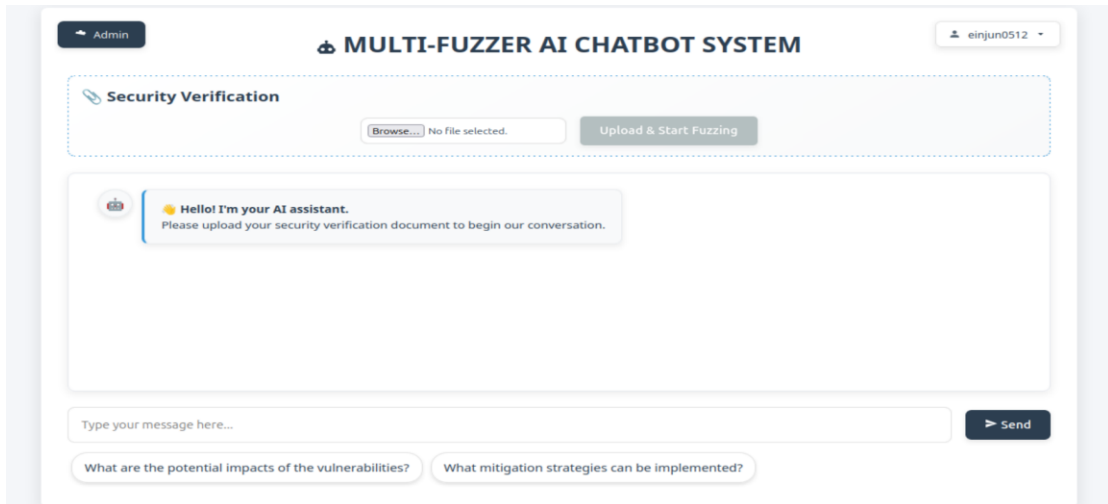
After entering their credentials, users are redirected to another page and required to enter a code sent to their authenticator (Google Authenticator) as part of the two-factor authentication process, ensuring that only authorized users can log in.

A screenshot of a 'Two-Factor Authentication' page. It features a title 'Two-Factor Authentication' with a shield icon and a subtitle 'Enter the 6-digit code from your authenticator app:'. Below this is a text input field containing the code '9 8 3 5 2 8'. A message 'Code expires in: 10s' is displayed. A blue 'Verify Code' button is positioned below the input field. At the bottom, a light blue box contains instructions: 'Open your authenticator app and enter the code shown there. The code changes every 30 seconds.'

**Figure 5.4.1.5:** Two-factor Authentication Page

### 5.4.2 Uploading a Binary

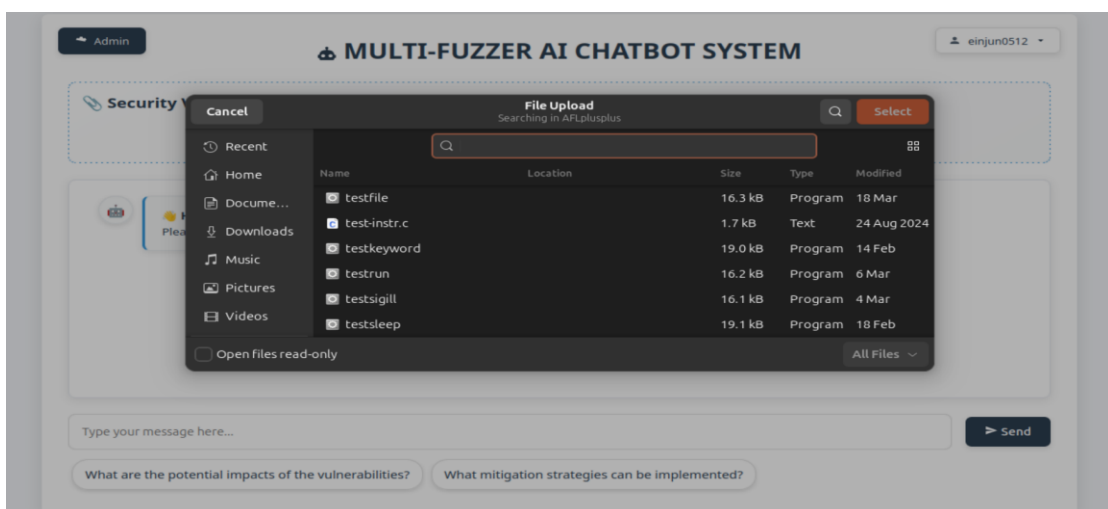
Once logged in, users can upload their binary files for fuzzing. The interface allows for easy selection and upload of target binaries.



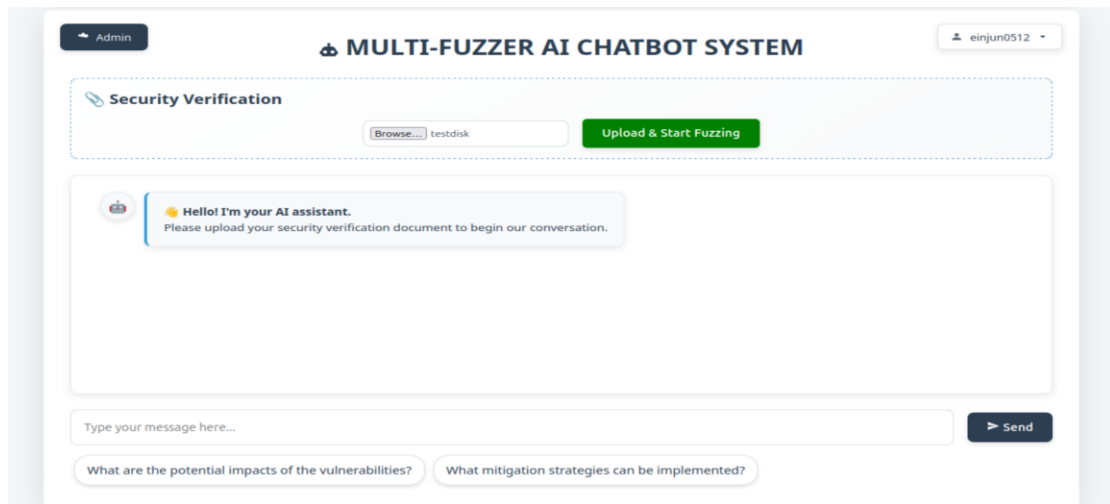
**Figure 5.4.2.1:** Multi-Fuzzer Chatbot System Main Interface

- Upload Process:

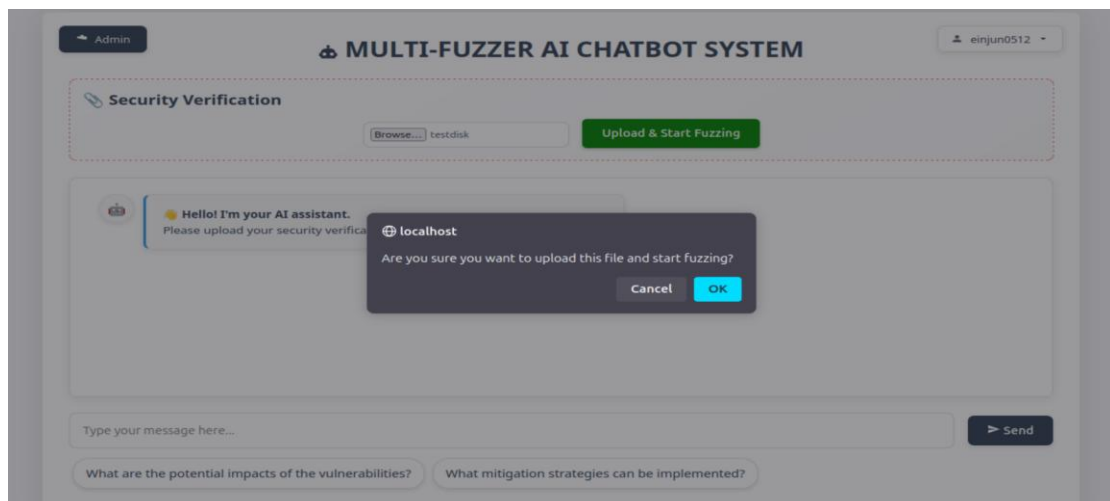
The user selects binary files (testdisk.elf and testfile.elf) from their local machine and uploads it via the web interface. The file is then transferred to the server, where it is stored in a designated directory for fuzzing.



**Figure 5.4.2.2:** Binary File Upload Process 1



**Figure 5.4.2.3:** Binary File Upload Process 2



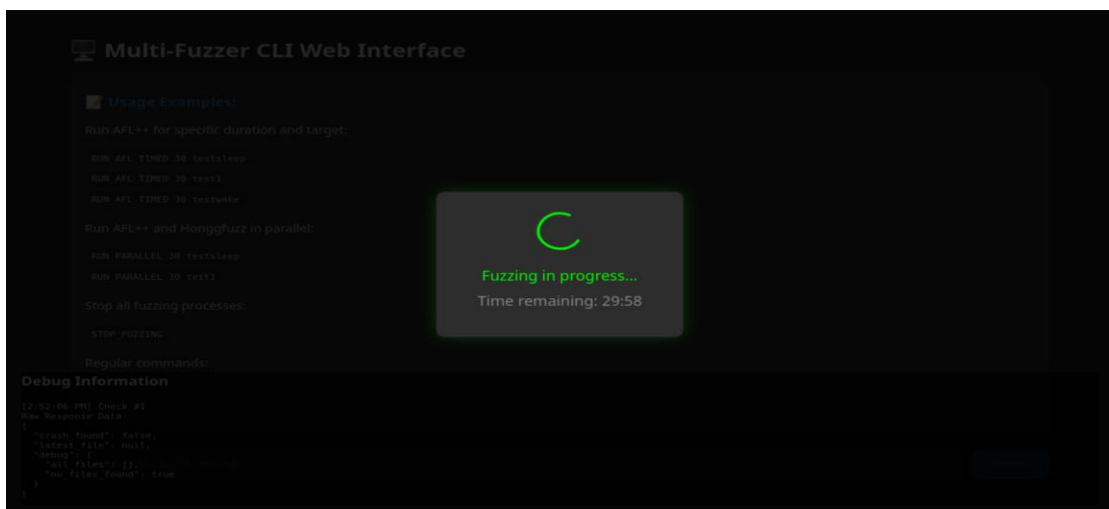
**Figure 5.4.2.4:** Binary File Upload Process 3

### 5.4.3 Fuzzing Begins

After the binary is uploaded, the fuzzing process begins. The system automatically triggers either AFL++ or Honggfuzz, depending on the configuration, and the fuzzing task runs in the background.

- **Fuzzing Execution:**

A terminal window shows the output of AFL++ or Honggfuzz as they execute the fuzzing process. The system captures any crashes or errors detected during this phase.



**Figure 5.4.3:** Fuzzing Process Begins

#### 5.4.4 Crash Detection and Report Combination

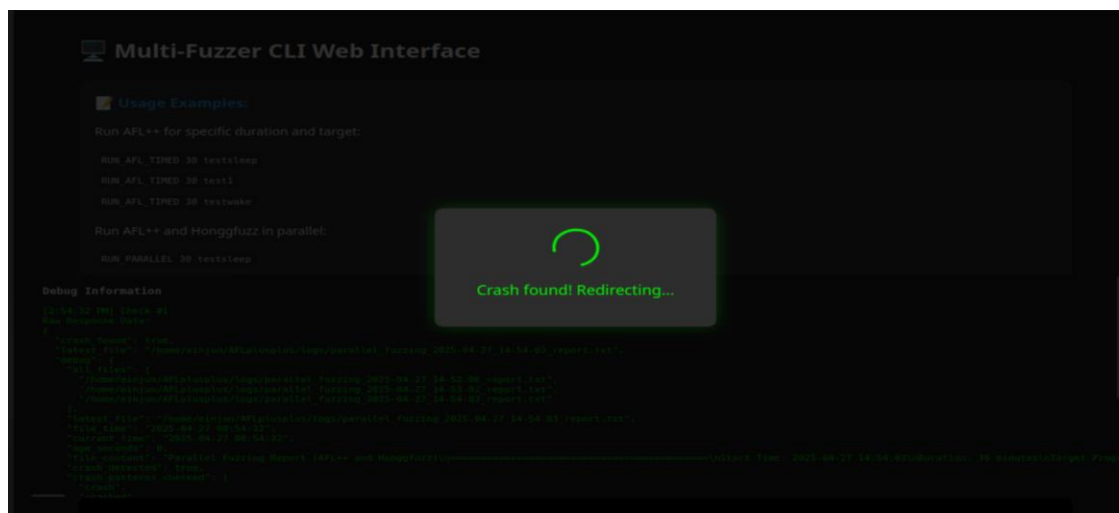
When a crash is detected, the system automatically stops fuzzing, processes the crash logs, and combines them into a single report. This report is then formatted and sent to the LLM (OpenAI GPT-3.5 Turbo API) for vulnerability analysis.

- **Crash Detection:**

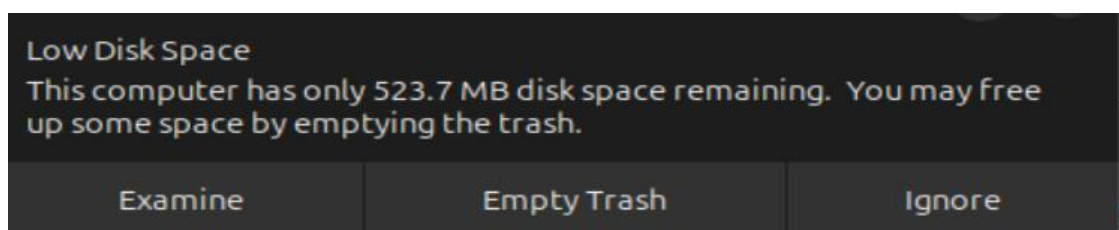
The fuzzing engine detects a crash, and the backend captures the crash log.

- **Report Combination:**

The system combines crash logs from AFL++ and Honggfuzz into a single structured report, ready to be sent to the LLM for analysis.



**Figure 5.4.4.1:** Crash Detected from Fuzzing Process



**Figure 5.4.4.2:** Low Disk Space Warning

Figure 5.4.4.2: Low Disk Space Warning illustrates the system's real-time response when a crash is detected during the fuzzing of the "testdisk" program, triggered by a disk space exhaustion attack. As the fuzzer injects malformed inputs, the program's



behavior leads to excessive disk usage, prompting the system to display a warning about critically low disk space. This alert not only signals the presence of a vulnerability but also helps prevent system instability by prompting immediate mitigation actions.

#### 5.4.5 CVE Report Generation with Mitigation Suggestion

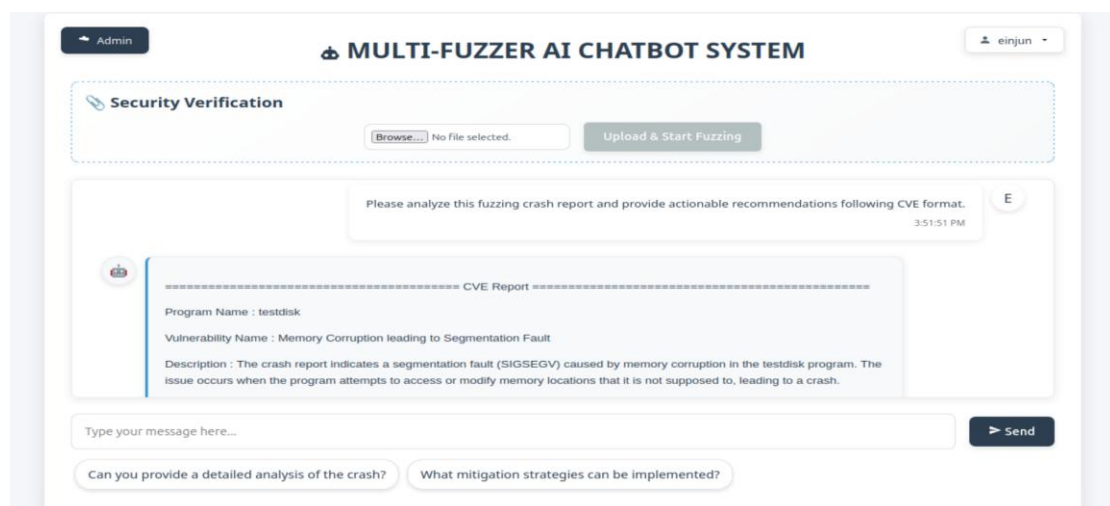
Once the data reaches the LLM, it processes the input and generates a CVE-style vulnerability report with suggested mitigations. This report is then displayed on the user interface.

- CVE Report Generation:

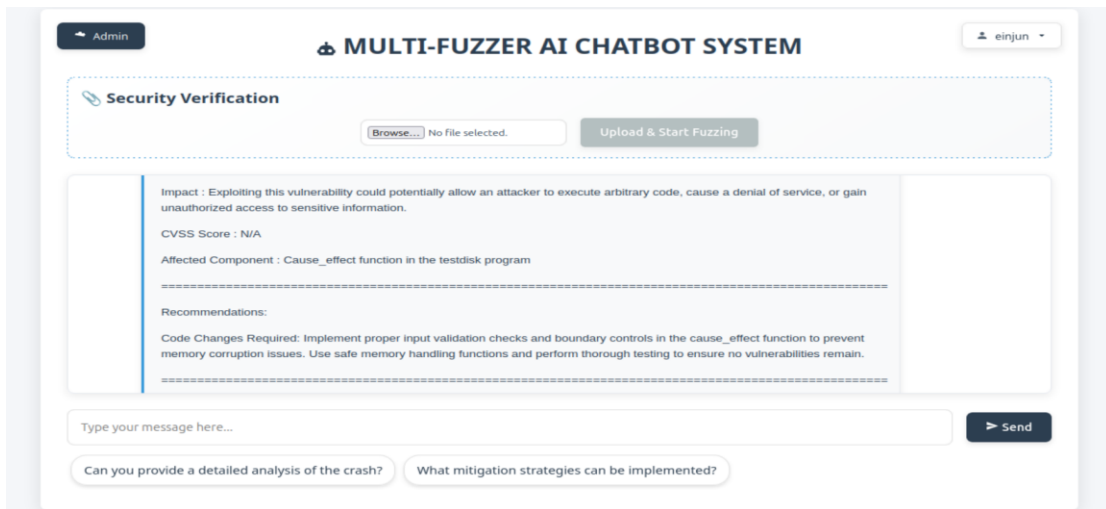
The LLM produces a detailed CVE report, including:

- Program Name
- Vulnerability Name
- Description
- Impact
- CVSS Score
- Affected Component

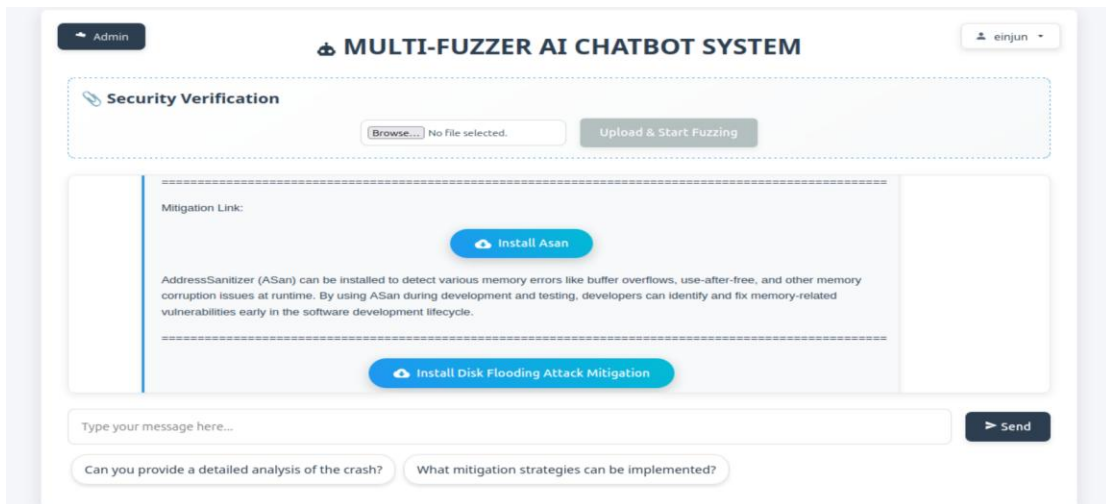
CVE Report generated for program: **testdisk.elf**



**Figure 5.4.5.1:** CVE Report for testdisk (Section 1) Generated by the LLM



**Figure 5.4.5.2:** CVE Report for testdisk (Section 2) Generated by the LLM



**Figure 5.4.5.3:** CVE Report for testdisk (Section 3) Generated by the LLM

CVE Report generated for program: **testfile.elf**

The screenshot shows the 'Security Verification' section of the 'MULTI-FUZZER AI CHATBOT SYSTEM'. At the top, there's a 'Browse...' button with the text 'No file selected.' and an 'Upload & Start Fuzzing' button. Below this, a message from the chatbot says: 'Please analyze this fuzzing crash report and provide actionable recommendations following CVE format. 3:40:34 PM'. The chatbot's response is a CVE report for 'testfile' with the following details:

- Program Name : testfile
- Vulnerability Name : Segmentation Fault in cause\_effect function

Below the report, there's a text input field with the placeholder 'Type your message here...' and a 'Send' button. Two suggested prompts are shown below the input field:

- Can you provide links to resources on these vulnerabilities?
- What mitigation strategies can be implemented?

**Figure 5.4.5.4:** CVE Report for testfile (Section 1) Generated by the LLM

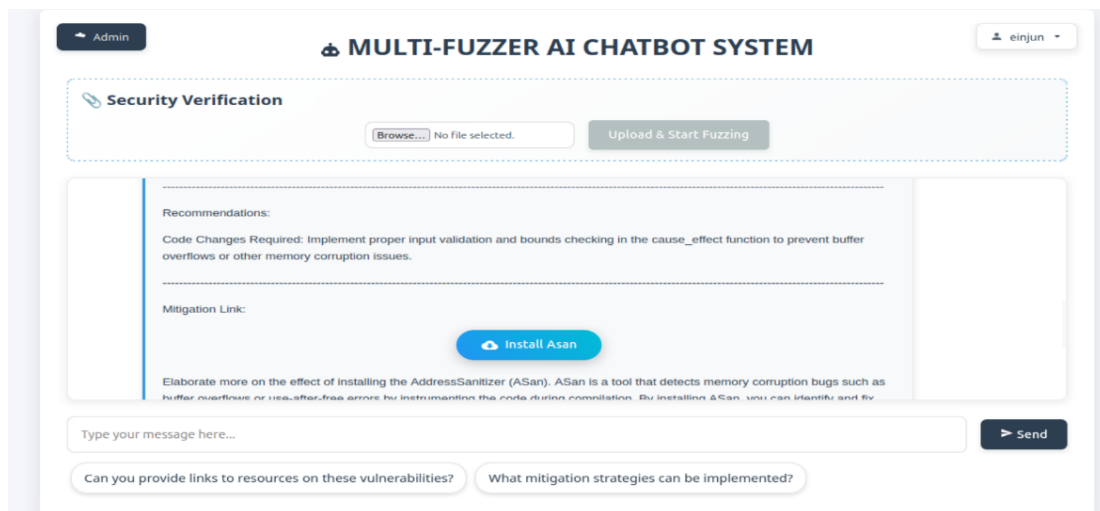
This screenshot shows a more detailed CVE report for 'testfile' generated by the LLM. The report includes the following information:

- Program Name : testfile
- Vulnerability Name : Segmentation Fault in cause\_effect function
- Description : The crash was triggered by a segmentation fault (SIGSEGV) in the cause\_effect function, leading to a program crash.
- Impact : Exploiting this vulnerability could potentially allow an attacker to execute arbitrary code, leading to denial of service or possibly remote code execution.
- CVSS Score : 7.8 (High)
- Affected Component : cause\_effect function in the testfile program

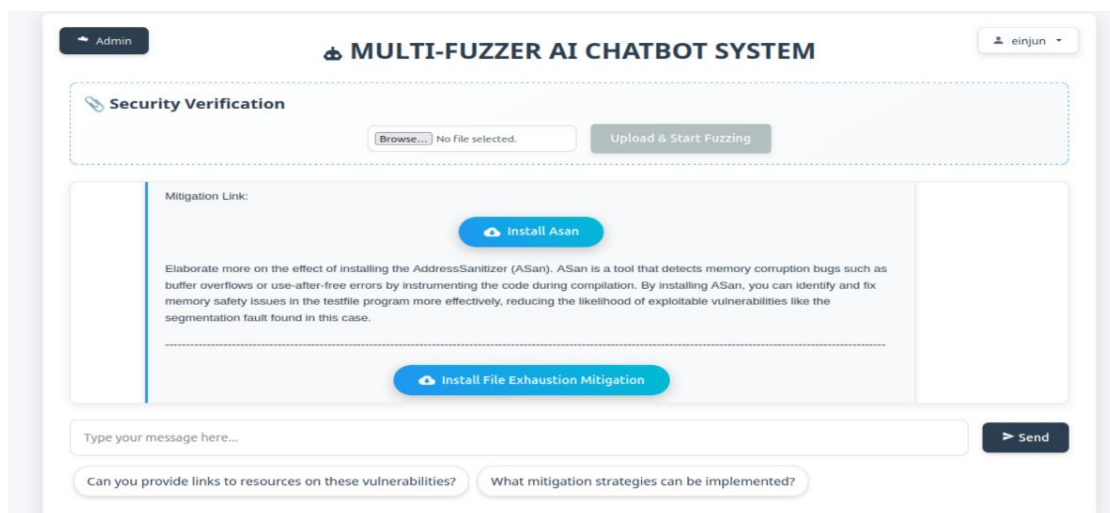
The interface also shows the 'Security Verification' section with 'Browse...' and 'Upload & Start Fuzzing' buttons. Below the report, there's a text input field with the placeholder 'Type your message here...' and a 'Send' button. Two suggested prompts are shown below the input field:

- Can you provide links to resources on these vulnerabilities?
- What mitigation strategies can be implemented?

**Figure 5.4.5.5:** CVE Report for testfile (Section 2) Generated by the LLM



**Figure 5.4.5.6:** CVE Report for testfile (Section 3) Generated by the LLM



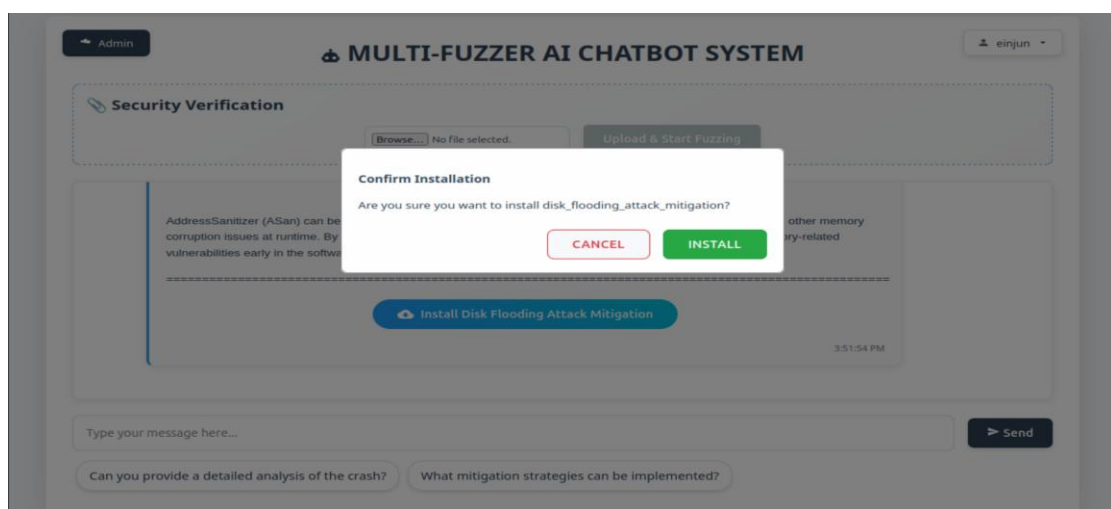
**Figure 5.4.5.7:** CVE Report for testfile (Section 4) Generated by the LLM

### 5.4.6 Mitigation Action Feedback

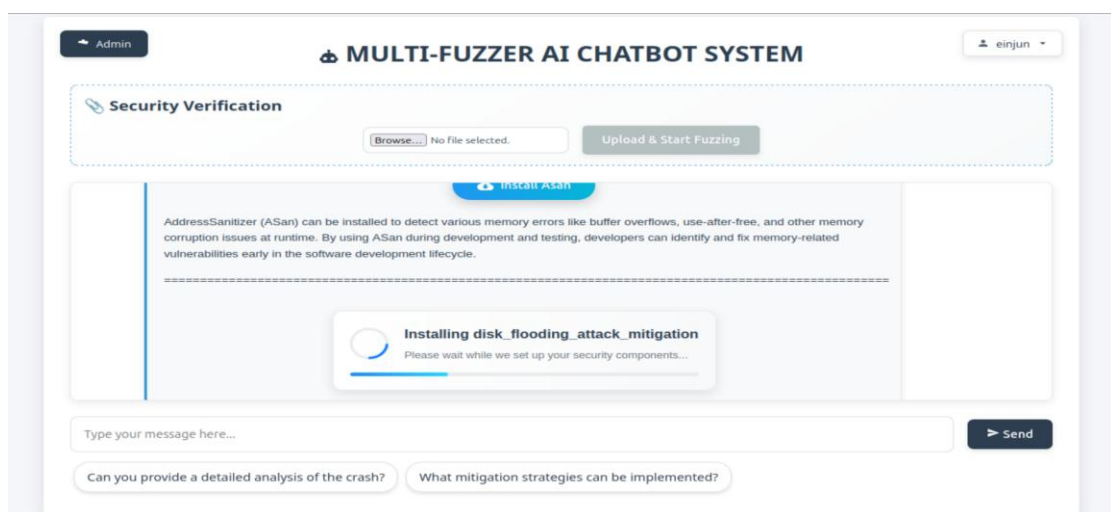
The user is presented with interactive mitigation options, which can be executed with the click of a button. Upon clicking the "Mitigate" button, the system triggers the corresponding mitigation script, which runs on the server.

- **Mitigation Execution:**

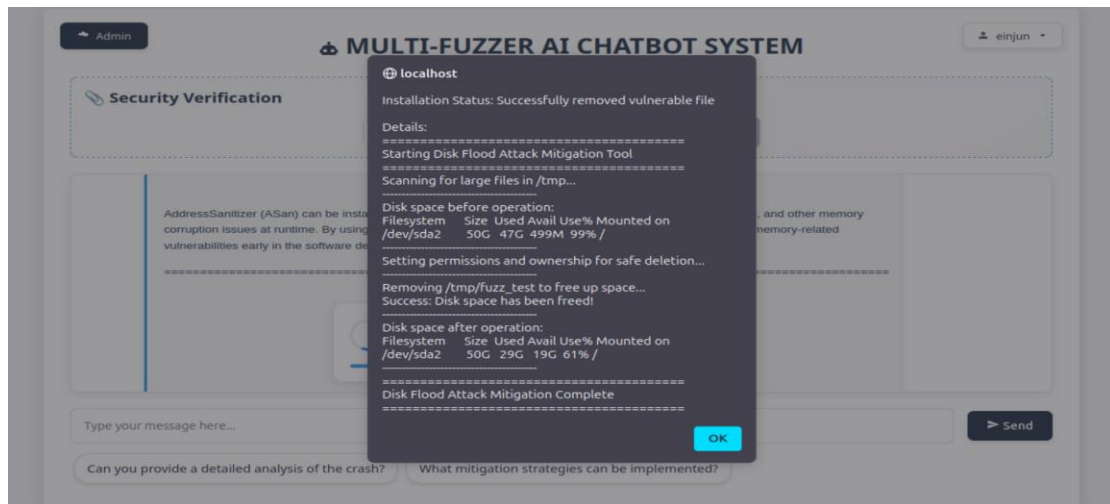
Once the user clicks the "Mitigate" button, a bash script runs to implement the suggested security measures. The user interface shows a "loading" status while the script executes and then displays the results.



**Figure 5.4.6.1:** Confirmation Pop-up for Mitigation Script Installation



**Figure 5.4.6.2:** Mitigation Script Installation Loading Status



**Figure 5.4.6.3: Mitigation Script Installation Successful**

### 5.4.7 Before/After Effect Visualization

After the mitigation script is executed, the system presents a clear visualization of the effects before and after the vulnerability was addressed. This feature is designed to help users understand how the mitigation improved the system's security or performance by comparing the state of the system before and after the intervention.

#### **Before/After Effect:**

The system displays a visual comparison of the specific state before and after the mitigation. For example, if a vulnerability involves disk space exploitation, such as inode exhaustion or disk flooding, the visualization will show how the disk space or inodes were impacted before and after the mitigation. The goal is to demonstrate how the mitigation script directly addresses the vulnerability and restores the system to a secure state.

Before/After Effect (testdisk.elf):

```
e1njun@e1njun-VirtualBox:~$ df -h
Filesystem      Size  Used Avail Use% Mounted on
tmpfs            452M  1.6M  450M   1% /run
/dev/sda2        50G   47G  500M  99% /
tmpfs            2.3G   22M   2.2G   1% /dev/shm
tmpfs            5.0M   8.0K   5.0M   1% /run/lock
tmpfs            2.3G   0     2.3G   0% /run/qemu
tmpfs            452M  140K  452M   1% /run/user/1000
```

**Figure 5.4.7.1:** Disk Flooding Attack (Before Effect)

This figure shows the disk space utilization before the mitigation script runs. The `df -h` command outputs the disk usage, highlighting any flooding or excessive consumption of disk space, which could lead to system instability or even denial of service (DoS) if not mitigated.

```
e1njun@e1njun-VirtualBox:~$ df -h
Filesystem      Size  Used Avail Use% Mounted on
tmpfs            452M  1.6M  450M   1% /run
/dev/sda2        50G   29G   19G  61% /
tmpfs            2.3G   0     2.3G   0% /dev/shm
tmpfs            5.0M   8.0K   5.0M   1% /run/lock
tmpfs            2.3G   0     2.3G   0% /run/qemu
tmpfs            452M  136K  452M   1% /run/user/1000
```

**Figure 5.4.7.2:** Disk Flooding Attack (After Effect)

After the mitigation script is applied, this figure shows how the disk usage has been brought back to normal, resolving the disk flooding vulnerability. The system is no longer overwhelmed by excessive disk usage, ensuring a secure environment and preventing potential system failures.

```
DISK_FLOODING_ATTACK_MITIGATION Status: Successfully removed vulnerable file

=====
Starting Disk Flood Attack Mitigation Tool
=====
Scanning for large files in /tmp...
-----
Disk space before operation:
Filesystem      Size  Used Avail Use% Mounted on
/dev/sda2        50G   47G  499M  99% /
-----
Setting permissions and ownership for safe deletion...
-----
Removing /tmp/fuzz_test to free up space...
Success: Disk space has been freed!
-----
Disk space after operation:
Filesystem      Size  Used Avail Use% Mounted on
/dev/sda2        50G   29G   19G  61% /
-----
Disk Flood Attack Mitigation Complete
=====
```

**Figure 5.4.7.3:** Before/After Effect of Disk Flooding Attack

Before/After Effect (testfile.elf):

```

winjun@winjun-VirtualBox:~$ stat -f /tmp
File: "/tmp"
ID: 5f58457cd30e8bf6 Namelen: 255      Type: ext2/ext3
Block size: 4096      Fundamental block size: 4096
Blocks: Total: 12854687 Free: 5411519 Available: 4804567
Inodes: Total: 3276800 Free: 2660116

```

**Figure 5.4.7.4:** File Inode Exhaustion (Before Effect)

This figure illustrates the state of the file system before the mitigation was applied, showing how inode exhaustion has led to reduced file system capacity. The command `stat -f /tmp` is used to display information about the file system, particularly the usage of inodes, showing a potentially insecure state with an excessive number of inodes consumed.

```

winjun@winjun-VirtualBox:~$ stat -f /tmp
File: "/tmp"
ID: 5f58457cd30e8bf6 Namelen: 255      Type: ext2/ext3
Block size: 4096      Fundamental block size: 4096
Blocks: Total: 12854687 Free: 5412475 Available: 4805523
Inodes: Total: 3276800 Free: 2785108

```

**Figure 5.4.7.5:** File Inode Exhaustion (After Effect)

After the mitigation script runs, this figure shows the resolved file system state. The inode usage has been reduced, bringing the system back to a secure state where inode consumption is within acceptable limits. This visualization shows how the mitigation script effectively restored the system's file handling to a stable, secure level.

```

FILE_EXHAUSTION_MITIGATION Status: Successfully cleaned up excessive files

=== INODE EXHAUSTION MITIGATION REPORT ===
[1] INITIAL SYSTEM STATE
-----
File: "/tmp"
ID: 5f58457cd30e8bf6 Namelen: 255      Type: ext2/ext3
Block size: 4096      Fundamental block size: 4096
Blocks: Total: 12854687 Free: 5411516 Available: 4804564
Inodes: Total: 3276800 Free: 2660115
[2] CLEANUP OPERATIONS
-----
Target Directory: /tmp/inode_flood/
Cleanup Command: /usr/bin/sudo rm -rf '/tmp/inode_flood/' 2>&1
Cleanup Results:
[3] FINAL SYSTEM STATE
-----
File: "/tmp"
ID: 5f58457cd30e8bf6 Namelen: 255      Type: ext2/ext3
Block size: 4096      Fundamental block size: 4096
Blocks: Total: 12854687 Free: 5412572 Available: 4805620
Inodes: Total: 3276800 Free: 2785116
[4] MITIGATION SUMMARY
-----
Status: SUCCESS - Files cleaned successfully
Directory Exists: No (Good)
Cleanup Return Code: 0
=== END OF REPORT ===
[5] INODE ANALYSIS
-----
Free Inodes Before: 2,660,115
Free Inodes After: 2,785,116
Inodes Freed: +125,001

```

**Figure 5.4.7.6:** Before/After Effect of File Inode Exhaustion



These consolidated figures provide a side-by-side view of the system before and after the mitigation, showing both inode exhaustion and disk flooding vulnerabilities and how they were addressed. The comparison helps users clearly visualize the impact of the mitigation and confirms that the vulnerabilities were successfully resolved.

#### 5.4.8 Further Questions Regarding the Uploaded Binary

After the fuzzing and initial analysis are completed, the system provides an option for users to ask additional questions regarding their uploaded binary and the associated vulnerabilities.

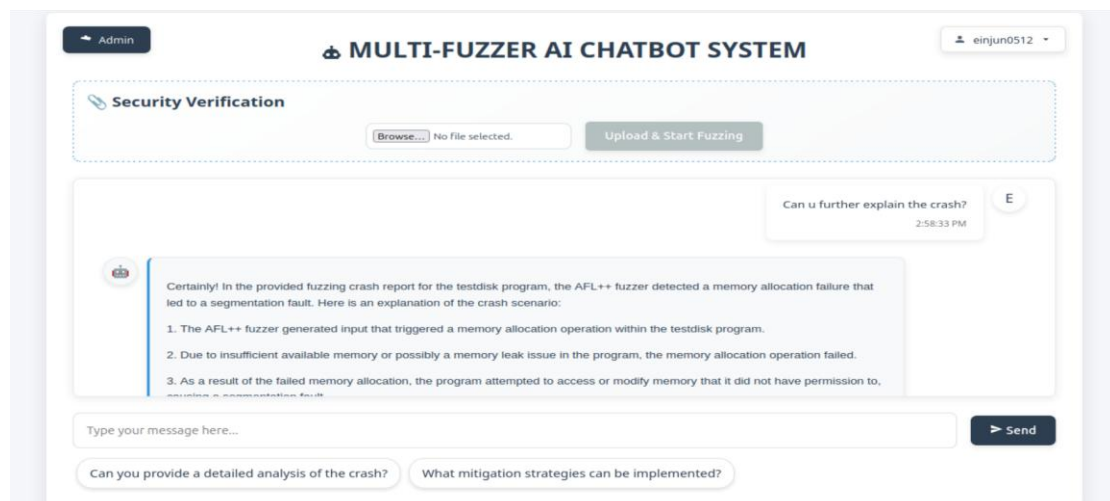
- **Question Interface:**

Users can input specific questions such as:

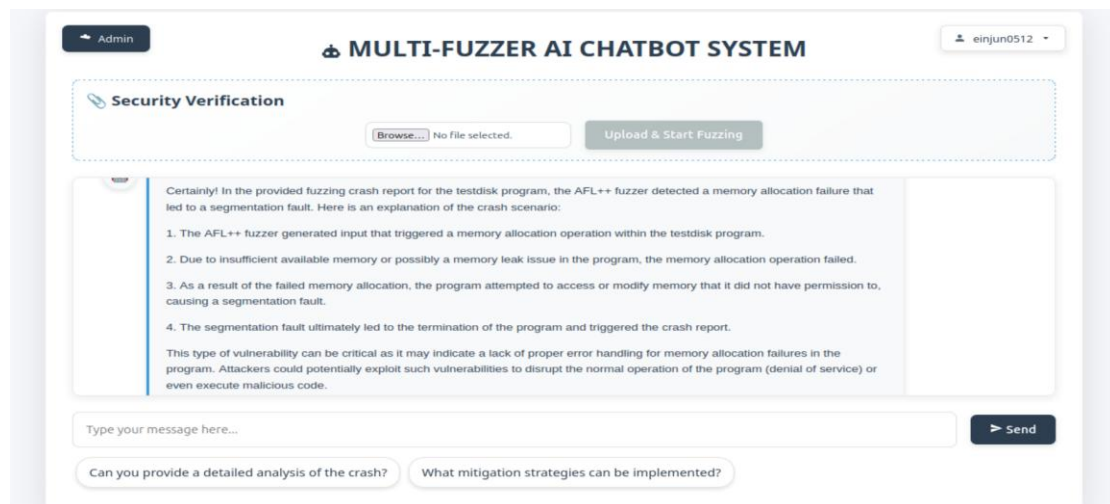
- "Is the vulnerability related to memory corruption?"
- "What kind of buffer overflow occurred?"
- "Can you suggest code-level patches for this?"

- **Response Generation:**

The system forwards these questions along with the previous crash report context to the LLM. The LLM provides answers or guidance based on the binary's fuzzing and crash data.



**Figure 5.4.8.1:** Further Question Input Box with Response (Section 1)



**Figure 5.4.8.2:** Further Question Input Box with Response (Section 2)

### 5.4.9 Admin Page with Authentication

An authenticated Admin Page is available for authorized users (admins or developers) to manage, debug, and interact with fuzzing tasks directly.

- **Authentication Gate:**

The admin page is protected and requires additional login credentials separate from the user login.

- **Admin Functionalities:**

Inside the Admin Panel, users can:

- View active fuzzing tasks.
- Manually input bash commands to debug the fuzzing processes.
- Force-stop or restart fuzzing jobs.
- View full raw logs from the fuzzer output.

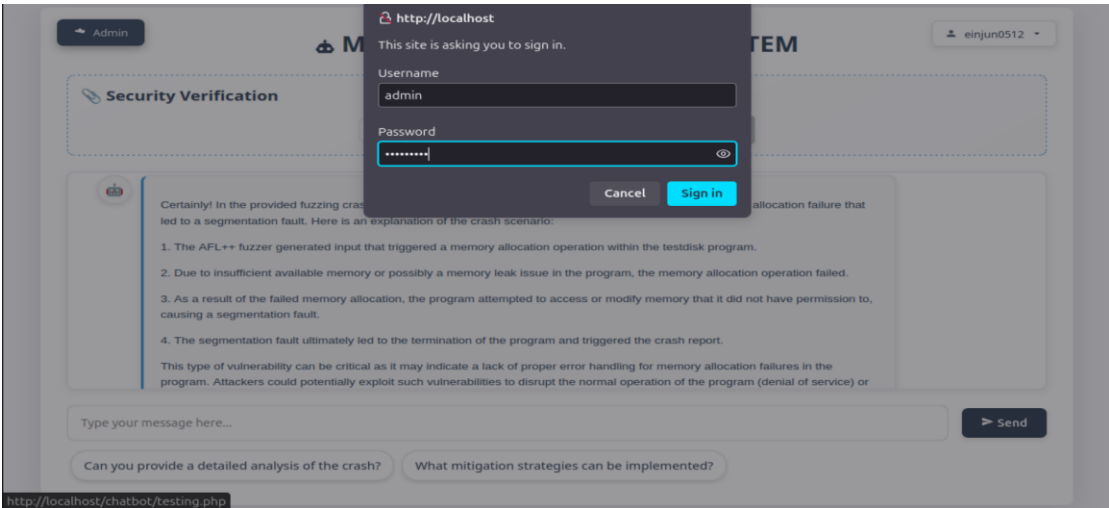


Figure 5.4.9.1: Admin Login Prompt



Figure 5.4.9.2: Debugging Dashboard with Command Input Area

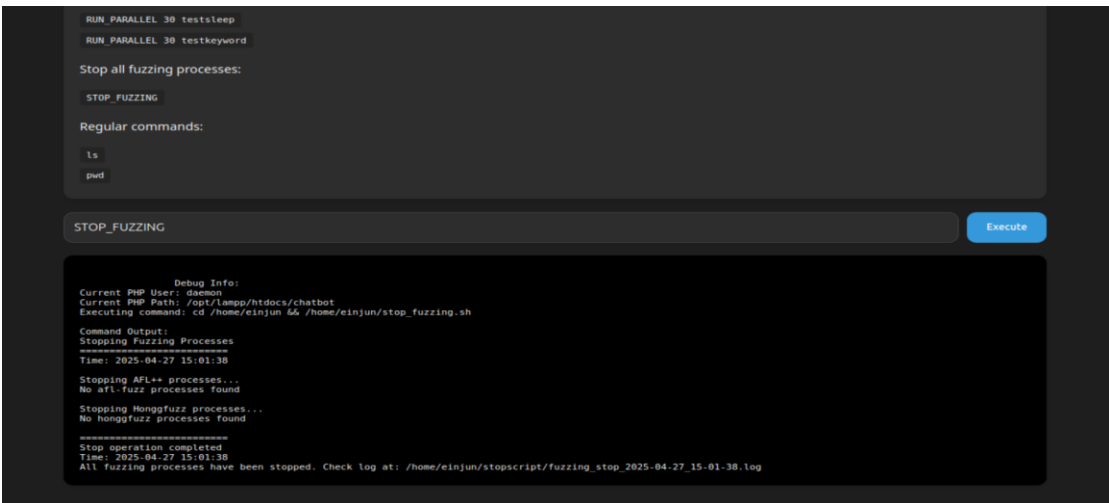


Figure 5.4.9.3: Force-stop Fuzzing Jobs

```

RUN_PARALLEL 30 testsleep

Debug Info:
Current PHP User: daemon
Current PHP Path: /usr/local/bin/php
Executing command: rm -rf /home/einjun/Desktop/outputthonggfuzz/* && cd /home/einjun && /home/einjun/run_parallel.sh 30 testsleep

Command Output:
Start time: 2025-04-27 15:02:24 bin: /home/einjun/AFLplusplus/testsleep, input: /home/einjun/Desktop/outputthonggfuzz/outputthonggfuzz_2025-04-27_15-02-24, persistent: false, stdin: true, mutation_rate: 5, timeout: 5, max_runs: 0, threads: 2, minimize: false, git_commit: ebacfad13e0766351149de8b3b54a1e6a017071
[21:50m] Entering phase 1/3: Dry Run
Launched new fuzzing thread, no. #0
Entered new fuzzing thread, no. #1
Entering phase 2/3: Switching to the Feedback Driven Mode
Entering phase 3/3: Dynamic Main (Feedback Driven Mode)
[0:36m] afl-fuzz++4.22a[0m based on afl by Michal Zalewski and a large online community
[1:02m+] [0mAFL++ is maintained by Marc "van Hauser" Heuse, Dominik Maier, Andrea Fioraldi and Heiko "hexcoder" Eißfeldt[0m
[1:02m+] [0mAFL++ is open source, get it at https://github.com/AFLplusplus/AFLplusplus[0m
[1:02m+] [0mNOTE: AFL++ -> v3 has changed defaults and behaviours - see README.md[0m
[1:02m+] [0mNo -M/-S set, autoconfiguring for "-S default"[0m
[1:04m+] [0mSetting up work...[0m
[1:02m+] [0mUsing exploration-based constant power schedule (EXPLORE)[0m
[1:02m+] [0mEnabled testcache with 50 MB[0m
[1:02m+] [0mGenerating fuzz data with a length of min=1 max=1048576[0m
[1:04m+] [0mChecking core pattern...[0m
[1:03m+] [1:07m]WARNING: [0mCould not check CPU scaling governor[0m
[1:02m+] [0mLooks like we're not running on a tty, so I'll be a bit less verbose.[0m
[1:02m+] [0mYou have 4 CPU cores and 7 runnable tasks (utilization: 175%).[0m
[1:03m+] [1:07m]WARNING: [0mSystem under apparent load, performance may be spotty.[0m
[1:04m+] [0mSetting up output directories...[0m
[1:02m+] [0mFound a free CPU core, try binding to #0.[0m
[1:04m+] [0mValidating target binary...[0m
[1:04m+] [0mScanning /home/einjun/AFLplusplus/input1...[0m
[1:04m+] [0mCreating hard links for all input files...[0m
[1:02m+] [0mLoaded a total of 1 seeds.[0m
[1:04m+] [0mNo auto-generated dictionary tokens to reuse.[0m
[1:04m+] [0mAttempting dry run with 'id:000000,time:0,execs:0,orig:seed5.txt'...[0m
[1:04m+] [0mJoining up the fork server...[0m
[1:02m+] [0mAll right - old fork server is up.[0m
[1:04m+] [0mTarget exe size: 65536[0m
[1:09m+] [0mLen = 5, map size = 30, exec speed = 7534 us, hash = 9509eb3be752f8d5
[0m[1:02m+] [0mAll test cases processed.[0m
[1:03m+] [1:07m]WARNING: [0mOnly one valid seed is present, auto-calculating the timeout is disabled![0m
[1:02m+] [0mHere are some useful stats:

[1:09m] Test case count : [0m1 favored, 0 variable, 0 ignored, 1 total
[1:09m] Bitmap range : [0m30 to 30 bits (average: 30.00 bits)
[1:09m] Exec timing : [0m7534 to 7534 us (average: 7534 us)
[0m
[1:04m+] [0m-t option specified. We'll use an exec timeout of 5000 ms.[0m
[1:02m+] [0mAll set and ready to roll![0m
[1:04m+] [0mFuzzing test case #0 (1 total, 0 crashes saved, state: started -), mode=explore, perf_score=100, weight=1, favorite=1, was_fuzzed=0, exec_us=7534, hits=0, map=30, execs=0, run time=0.00.00.00) [0m
Crash: saved as /home/einjun/Desktop/outputthonggfuzz/SIGSEGV_PC_55555555318_STACK_cbb542e7a.CODE.1.ADDR.0.INSTR.mov 'eax, (%rdx).fuzz'
Crash (dup): /home/einjun/Desktop/outputthonggfuzz/SIGSEGV_PC_55555555318_STACK_cbb542e7a.CODE.1.ADDR.0.INSTR.mov 'eax, (%rdx).fuzz' already exists, skipping
Crash (dup): /home/einjun/Desktop/outputthonggfuzz/SIGSEGV_PC_55555555318_STACK_cbb542e7a.CODE.1.ADDR.0.INSTR.mov 'eax, (%rdx).fuzz' already exists, skipping
Crash (dup): /home/einjun/Desktop/outputthonggfuzz/SIGSEGV_PC_55555555318_STACK_cbb542e7a.CODE.1.ADDR.0.INSTR.mov 'eax, (%rdx).fuzz' already exists, skipping
Crash (dup): /home/einjun/Desktop/outputthonggfuzz/SIGSEGV_PC_55555555318_STACK_cbb542e7a.CODE.1.ADDR.0.INSTR.mov 'eax, (%rdx).fuzz' already exists, skipping
Crash (dup): /home/einjun/Desktop/outputthonggfuzz/SIGSEGV_PC_55555555318_STACK_cbb542e7a.CODE.1.ADDR.0.INSTR.mov 'eax, (%rdx).fuzz' already exists, skipping
Crash (dup): /home/einjun/Desktop/outputthonggfuzz/SIGSEGV_PC_55555555318_STACK_cbb542e7a.CODE.1.ADDR.0.INSTR.mov 'eax, (%rdx).fuzz' already exists, skipping
Crashes found! Check report at: /home/einjun/AFLplusplus/logs/parallel_fuzzing_2025-04-27_15-02-24_report.txt

```

Figure 5.4.9.4: Debugging Fuzzing Process

### 5.4.10 Account Settings Page

The system provides a comprehensive Account Settings Page, allowing users to manage security features like password changes and 2FA setup.

- **Password Change:**

Users can update their account password by entering:

- Current password
- New password
- Confirmation of the new password

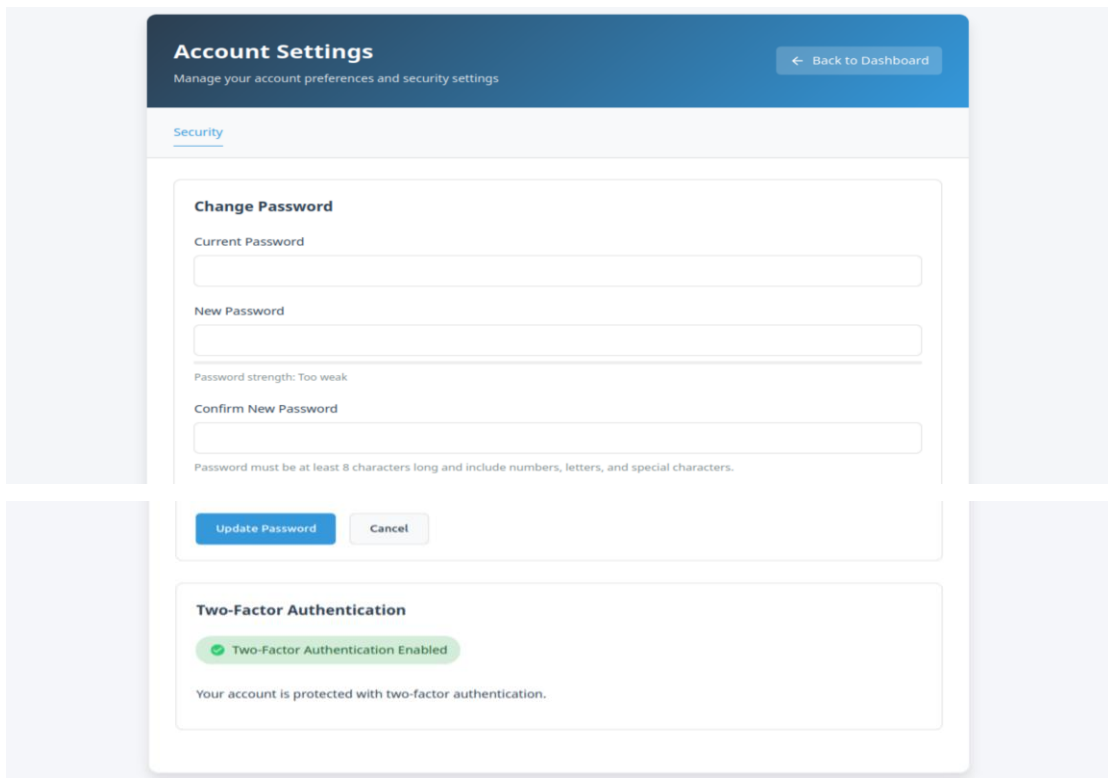
- **2FA Setup (Authenticator App Integration):**

Users can enable 2FA by:

- Scanning a QR Code using an Authenticator App (e.g., Google Authenticator, Microsoft Authenticator).
- Entering the 6-digit verification code generated by the app.
- Upon successful entry, the system will confirm that 2FA has been enabled successfully.

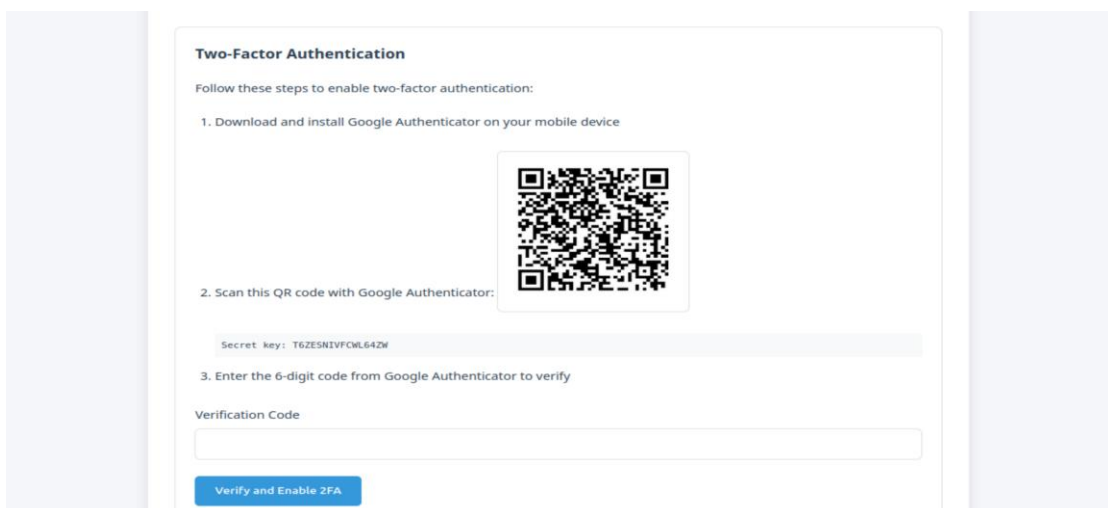
- **2FA Status:**

The system also displays whether 2FA is currently enabled or disabled for the account.



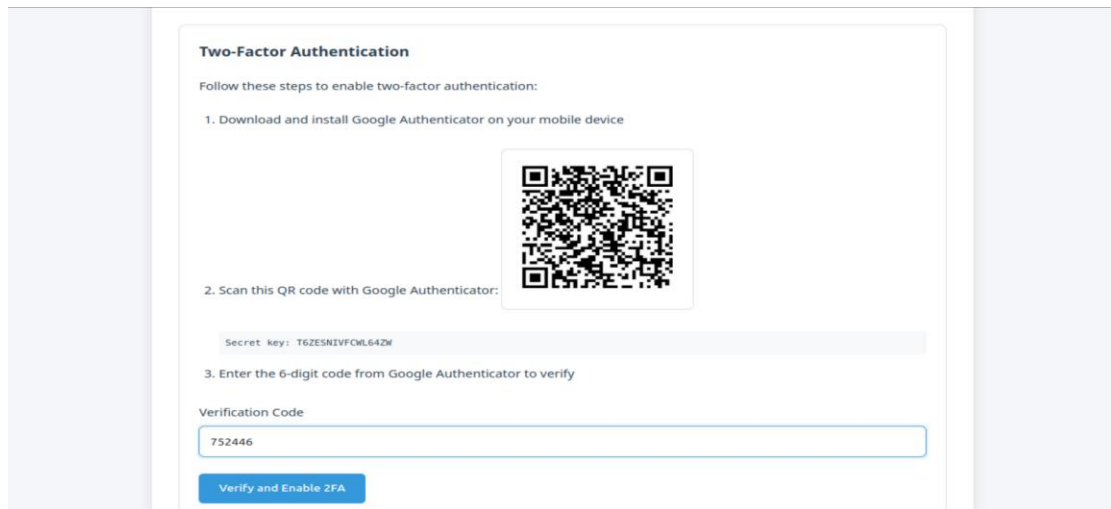
The screenshot shows the 'Account Settings' page with a blue header. Below the header, there's a 'Security' tab. The 'Change Password' section has three input fields: 'Current Password', 'New Password', and 'Confirm New Password'. A message below the 'New Password' field states 'Password strength: Too weak' and 'Password must be at least 8 characters long and include numbers, letters, and special characters.' At the bottom of this section are 'Update Password' and 'Cancel' buttons. The 'Two-Factor Authentication' section shows a green checkmark and the text 'Two-Factor Authentication Enabled' and 'Your account is protected with two-factor authentication.'

**Figure 5.4.10.1:** Account Setting Page with 2FA Enabled



The screenshot shows the 'Two-Factor Authentication' activation process. It includes a list of steps: 1. Download and install Google Authenticator on your mobile device. 2. Scan this QR code with Google Authenticator: (A QR code is displayed). 3. Enter the 6-digit code from Google Authenticator to verify. Below the QR code, a 'Secret key' is shown: 'T6ZESNIVFCWL64ZW'. At the bottom, there is a 'Verification Code' input field and a 'Verify and Enable 2FA' button.

**Figure 5.4.10.2:** Account Setting Page: 2FA Activation Process (Section 1)



**Two-Factor Authentication**

Follow these steps to enable two-factor authentication:

1. Download and install Google Authenticator on your mobile device
2. Scan this QR code with Google Authenticator:

Secret key: T6ZESNIVFCML64ZM

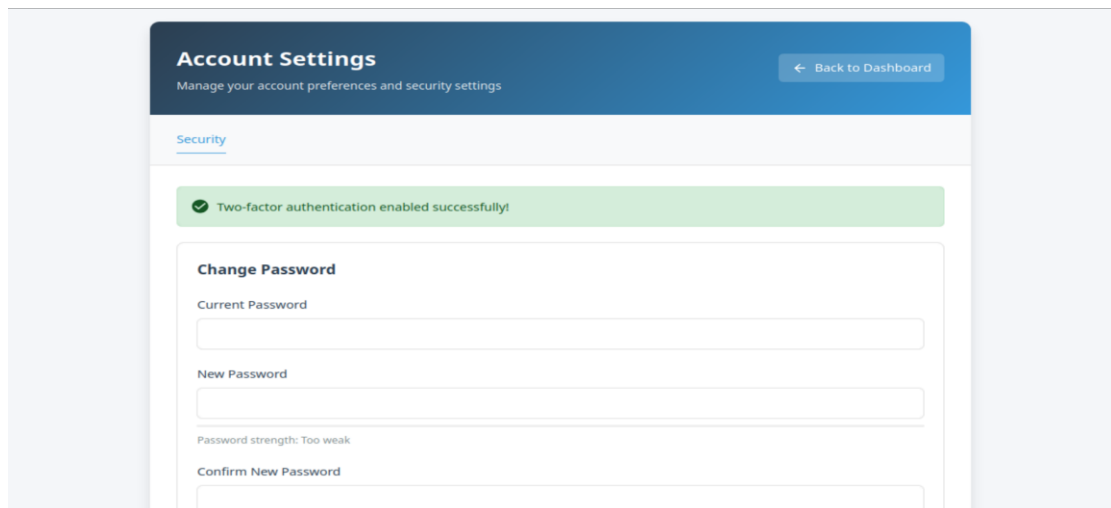
3. Enter the 6-digit code from Google Authenticator to verify

Verification Code

752446

Verify and Enable 2FA

**Figure 5.4.10.3:** Account Setting Page: 2FA Activation Process (Section 2)



**Account Settings**

Manage your account preferences and security settings

[← Back to Dashboard](#)

Security

✓ Two-factor authentication enabled successfully!

**Change Password**

Current Password

New Password

Password strength: Too weak

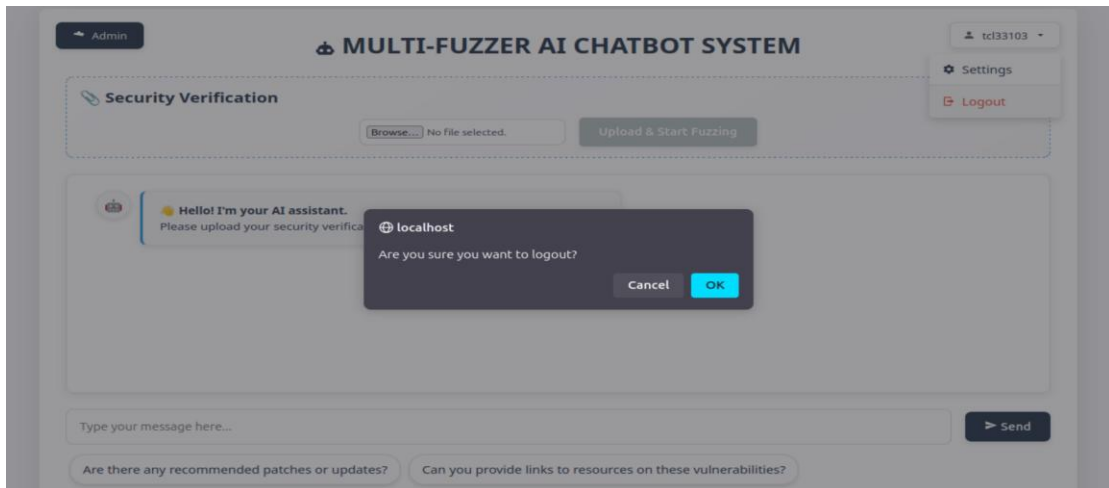
Confirm New Password

**Figure 5.4.10.4:** Account Setting Page: 2FA Activated Successfully

### 5.4.11 Logout Functionality

A Logout Button is provided across the user interface for security and session management purposes.

- **Functionality:**
  - When clicked, the user session is immediately destroyed.
  - All authentication tokens, including 2FA tokens, are invalidated.
  - The user is redirected securely to the Login Page.
- **Security:**
  - Proper session destruction prevents unauthorized reuse of login sessions, ensuring user data remains protected even after logout.



**Figure 5.4.11:** Logout Confirmation Prompt

## 5.5 Implementation Issues and Challenges

### 5.5.1 Fuzzer Not Crashing Consistently

- **Problem and Impact:**

Despite running AFL++ and Honggfuzz, there were instances when these fuzzers failed to generate crashes for certain binaries, even though vulnerabilities were suspected. The fuzzers did not always produce crashes consistently, particularly with binaries from the picoCTF challenge set, where certain files contained deep vulnerabilities that required extensive time to expose. As fuzzing results are crucial for identifying vulnerabilities, this inconsistency in crash generation caused gaps in vulnerability assessment. In some cases, vulnerabilities went undetected, which reduced the effectiveness of the system in evaluating the security of the provided binaries.

- **Solution:**

To tackle this issue, I adjusted the fuzzing configurations, such as increasing the allotted time for fuzzing sessions and diversifying the seed files used in the fuzzing process. The seed files were carefully selected to include a wider range of edge cases to ensure that fuzzing would cover more potential vulnerability scenarios. Additionally, more detailed logging mechanisms were implemented to track fuzzing progress and identify why certain binaries did not generate crashes, allowing for further refinement and troubleshooting of the fuzzing process.

### 5.5.2 Integration of Bash Scripts with PHP Frontend

- **Problem and Impact:**

The integration between Bash scripts (responsible for controlling the fuzzing process and generating reports) and the PHP frontend posed several challenges. Specifically, the PHP server would sometimes fail to execute Bash commands, or it would hang while waiting for the output. This was particularly problematic in the context of real-time fuzzing feedback, as users could not see the progress or results of fuzzing immediately. The lack of smooth communication between the frontend and the backend caused delays in displaying results to users and



disrupted the seamless flow of the fuzzing process, ultimately degrading the user experience.

- **Solution:**

To address this, I optimized the way PHP executed Bash scripts using the `exec()` function, ensuring that Bash commands could be executed properly. I also made sure that the PHP server had the correct permissions to run these scripts. Additionally, I incorporated robust error handling and logging in the PHP code to track any issues that occurred during script execution. This improved the overall reliability of the system, ensuring that fuzzing tasks could be executed without interruption and that users received real-time updates on the progress and results of the fuzzing process.

### 5.5.3 Handling Different Types of Vulnerabilities

- **Problem and Impact:**

The fuzzing process often revealed various types of vulnerabilities, including buffer overflows, memory leaks, and input validation errors. However, these vulnerabilities were not always handled consistently across different fuzzing engines. For instance, some engines would produce detailed crash reports for specific types of vulnerabilities, while others would miss certain edge cases. The inconsistent identification and categorization of vulnerabilities caused confusion in the reporting process and sometimes resulted in incomplete mitigation suggestions. This inconsistency hindered the system's ability to deliver accurate and actionable vulnerability reports to users.

- **Solution:**

To resolve this, I implemented a categorization system that classified vulnerabilities and target binaries based on predefined tags. This allowed for more consistent handling and reporting of vulnerabilities, ensuring that each vulnerability type had a corresponding mitigation strategy. Additionally, the fuzzing engines were fine-tuned to better target specific types of vulnerabilities, which helped improve the precision of the fuzzing results and the quality of the mitigation recommendations.

#### 5.5.4 Difficulty in Implementing Two-Factor Authentication

- **Problem and Impact:**

Integrating Two-Factor Authentication (2FA) with Google Authenticator presented significant challenges. The process of generating and verifying QR codes, as well as securely managing the user input, was complex. Additionally, there were some security concerns related to ensuring that the 2FA process was seamless and didn't interfere with the overall user experience. Users also had difficulty setting up 2FA, leading to a higher likelihood of errors during configuration and setup.

- **Solution:**

To overcome these challenges, I streamlined the 2FA process by simplifying the generation and scanning of QR codes using the Google Authenticator app. Detailed error handling and user-friendly prompts were incorporated to guide users through the setup process, ensuring that they could complete 2FA activation without issues. This enhancement improved security while maintaining ease of use for users, ensuring that 2FA was both effective and non-intrusive.

#### 5.5.5 File Permissions and Sudo Requirement

- **Problem and Impact:**

Some files and scripts required elevated permissions (sudo) to execute, which caused issues when trying to integrate these components with the web frontend. Web browsers generally restrict the ability to run scripts with elevated privileges, creating a conflict between the necessary permissions for the backend scripts and the security restrictions of the browser. This prevented the fuzzing process from running as expected, particularly when certain scripts were needed to access restricted resources.

- **Solution:**

I resolved this issue by adjusting file permissions to ensure that only the necessary scripts had elevated privileges and by minimizing the need for sudo access. Additionally, I ensured that scripts requiring higher privileges were executed outside the web interface, reducing the risk of security vulnerabilities.

This allowed the system to function properly while maintaining the necessary security controls.

### 5.5.6 Inconsistent Responses for User Queries

- **Problem and Impact:**

Initially, the system provided responses based on the immediate conversation history, which sometimes resulted in inconsistent or incomplete answers. When users inquired about previously addressed topics, the system was unable to generate relevant responses, leading to confusion and frustration.

- **Solution:**

A conversation history feature was introduced, allowing the system to reference past interactions. This enabled the system to provide consistent and accurate responses based on the full history of the conversation, improving user satisfaction and the overall functionality of the chatbot.

### 5.6 Concluding Remark

This section outlined the development and implementation of the Multi-Fuzzer Automated Vulnerability Assessment System, which combines fuzzing engines, a vulnerability analysis pipeline, and an intuitive user interface to automate the process of detecting and mitigating security vulnerabilities in uploaded binaries. Overall, the system has effectively achieved its core objectives of automating vulnerability detection and providing actionable insights, all while maintaining an intuitive and user-friendly experience.

#### **Achievement of System Objectives:**

The implementation of this system has successfully met its primary goals. The integration of multiple fuzzing engines (AFL++ and Honggfuzz) ensures that a wide range of vulnerabilities can be tested across different types of binaries. The use of OpenAI's GPT-3.5 Turbo LLM allowed us to automate the analysis of crash reports, generate CVE-style reports, and suggest mitigation strategies. This level of automation minimizes user intervention while providing highly valuable insights into the vulnerabilities detected.

The user interface (UI) offers a simple, streamlined process for uploading files, monitoring fuzzing progress, and receiving results, which is ideal for developers and security professionals. With clear visual feedback and interactive mitigation suggestions, the system simplifies the complex process of vulnerability assessment and response.

#### **Areas for Future Improvement:**

While the system has performed well within its current scope, there are a few areas where improvements could be made to enhance functionality in future versions:

1. **Training a Custom LLM for Security Needs:** Rather than relying on a pretrained LLM like GPT-3.5, future versions of the system could explore training a custom LLM specifically designed for security vulnerabilities. This LLM would be trained using a curated dataset of security-related crash reports,

CVE data, and known mitigation strategies. By using a model tailored to security needs, we can expect more accurate and contextually relevant analysis, without the token limit constraints associated with current LLMs. This shift would also improve processing time and eliminate the need for pre-processing or chunking of crash reports.

2. **Enhanced Database Integration:** Although the current system handles report storage and processing, future versions could incorporate a more sophisticated database for storing detailed vulnerability data. This would include information like CVSS scores, affected components, and historical vulnerability data, which would enhance the system's ability to track, analyze, and report on vulnerabilities over time. A more robust database integration could also support advanced query capabilities and better scalability.
3. **Advanced Reporting and Visualization:** The current reporting system generates CVE-style reports, but future improvements could offer more advanced visualizations of vulnerability trends, system performance, and deeper insights into individual vulnerabilities. This could include graphical representations of affected code areas, risk levels, and detailed mitigation steps.
4. **Broader Fuzzing Engine Support:** Currently, the system supports AFL++ and Honggfuzz. Future updates could introduce additional fuzzing engines to extend the range of vulnerabilities detected, allowing the system to assess a wider variety of binaries and software environments.

### **Smoothness of User Experience and Automation Level Achieved:**

The system offers a smooth and intuitive user experience, guiding users through each step of the vulnerability assessment process. File uploads are simple, fuzzing progress is tracked in real time, and results are displayed in a clear, actionable format. Users are also presented with interactive mitigation suggestions that allow them to take immediate action.

In terms of automation, the system excels in streamlining complex tasks. The fuzzing process, crash detection, report generation, and mitigation suggestion features all run

automatically, with minimal user input required. However, there are still opportunities for further automation, particularly in the management of crash reports and the handling of the fuzzing process. Future versions could focus on enhancing these automated workflows, particularly around handling large crash reports and improving database integration.

## CHAPTER 6

### System Evaluation and Discussion

#### 6.1 System Testing and Performance Metrics

To ensure the developed system meets its intended functionality and performance requirements, extensive testing was conducted across multiple dimensions, including functionality, performance, usability, and system sturdiness. The primary goal of system testing was to validate the integration of multi-fuzzing engines, crash report generation, LLM-based vulnerability assessment, and user interface interactions in a real-world environment.

##### Testing Methods

The following testing strategies were employed:

- **Functional Testing:**

Functional testing focused on verifying that each module within the system, such as file uploading, fuzzing initiation, crash aggregation, CVE-like reporting, and mitigation operations correctly and as intended. Each user action, from file upload to receiving a completed vulnerability report, was thoroughly tested to ensure end-to-end workflow reliability.

- **Performance Testing:**

The system's performance was evaluated based on fuzzing speed, crash detection rates, system response times, and resource utilization (CPU and memory) during concurrent fuzzing operations. Both normal and stress scenarios were simulated to measure the system's stability under load.

- **Usability Testing:**

The user interface was tested for intuitiveness and responsiveness. Focus was placed on the file upload process, access to crash reports, system control

features, and account management (e.g., 2FA setup). Smoothness of user navigation and task completion time were considered as usability metrics.

- **Stress and Scalability Testing:**

The system was subjected to heavy testing loads, such as running multiple fuzzing sessions simultaneously, processing large binary files, and generating a high volume of crash reports. This helped assess the system's capability to maintain performance without degradation under extreme conditions.

- **Security Testing:**

Testing was also conducted to verify the robustness of security features, including login authentication, admin access controls, and two-factor authentication (2FA) setup. Attempts were made to simulate unauthorized access scenarios to validate the system's protection mechanisms.

### Performance Metrics

The following key performance metrics were used to evaluate the system:

- **Fuzzing Speed:**

Measured by the number of test cases generated and executed per minute by the fuzzers (AFL++ and Honggfuzz). This metric indicated the system's efficiency in exploring vulnerabilities within uploaded binaries.

- **Crash Detection Rate:**

Calculated by the number of crashes detected relative to the total fuzzing time. A higher detection rate implied better effectiveness in exposing vulnerabilities.



- **Resource Utilization:**

Monitored CPU and memory consumption during active fuzzing sessions. Acceptable thresholds were defined to ensure system responsiveness and avoid overload, especially when running concurrent fuzzers.

- **Response Time for CVE Report Generation:**

The time taken from the moment a crash was detected until a structured CVE-like vulnerability report was generated and displayed to the user.

- **Error Rate and Recovery:**

Monitored how frequently system errors occurred and how effectively the system recovered from such errors through automated reprocessing or user prompts.

- **Usability Metrics:**

Evaluated based on user feedback regarding the ease of file uploads, crash report accessibility, and account management.

### Quality Metrics:

The **number of unique vulnerabilities** metric tracks how many distinct vulnerabilities are discovered during the testing process. This metric helps evaluate the effectiveness of the fuzzing approach in identifying new and previously undetected issues. A higher count of unique vulnerabilities suggests a more thorough and effective testing process.

**Severity of vulnerabilities** assesses the criticality and potential impact of the identified issues. It classifies vulnerabilities based on their risk levels, such as high, medium, or low. Prioritizing high-severity vulnerabilities ensures that the most critical issues are addressed first, enhancing the overall security posture and risk management of the application. The tables below illustrate the **different types of vulnerabilities associated with specific signals that were detected** during the fuzzing process. Each

signal corresponds to a particular type of error or exception in the system, which, when triggered, can expose unique vulnerabilities in the software.

**Table 6.1.1: SIGSEGV Vulnerabilities**

Signal 11 (SIGSEGV - Segmentation Fault)	
Vulnerability Types	Severity of vulnerabilities
Buffer Overflow	High
Null Pointer Dereference	Medium
Use-After-Free	High
Stack Overflow	Low
Dangling Pointer Access	Medium
Unaligned Memory Access	Low
Memory Corruption	Low

The **SIGSEGV** (Segmentation Fault) signal occurs when a program attempts to access an invalid memory location. This is a critical signal as it often leads to serious vulnerabilities:

- **Buffer Overflow** and **Use-After-Free** are high-severity vulnerabilities that can allow attackers to execute arbitrary code or manipulate memory, leading to complete system compromise.
- **Null Pointer Dereference** and **Stack Overflow** are medium-severity issues that can result in crashes or unintended behavior.
- **Dangling Pointer Access** and **Memory Corruption** are lower-severity vulnerabilities that may lead to subtle bugs but can also escalate under certain conditions.

**Table 6.1.2: SIGALRM Vulnerabilities**

Signal 14 (SIGALRM - Alarm Clock)	
Vulnerability Types	Severity of vulnerabilities
Denial of Service (DoS)	High
Race Conditions	Medium
Timeout Vulnerabilities	Medium
Resource Exhaustion	Low

The **SIGALRM** signal is generated when a timer set by the `alarm()` function expires.

This signal can expose vulnerabilities related to time management and resource usage:

- **Denial of Service (DoS)** vulnerabilities are rated as high severity because they can render a service unavailable, disrupting operations.
- **Race Conditions** and **Timeout Vulnerabilities** are medium-severity vulnerabilities that can cause unpredictable behavior or allow attackers to exploit timing issues.
- **Resource Exhaustion** is a low-severity vulnerability, but if left unaddressed, it can degrade system performance over time.

**Table 6.1.3:** SIGABRT Vulnerabilities

Signal 6 (SIGABRT - Abort)	
Vulnerability Types	Severity of vulnerabilities
Assertion Failures	High
Memory Leaks	Low
Inconsistent State	Medium
Denial of Service (DoS)	High

The **SIGABRT** signal is sent by the `abort()` function, typically indicating an assertion failure or an abnormal termination of the program:

- **Assertion Failures** and **Denial of Service (DoS)** vulnerabilities are high-severity issues, as they can lead to abrupt terminations or crashes, potentially allowing further exploits.
- **Memory Leaks** are low-severity but can accumulate over time, leading to degraded performance.
- **Inconsistent State** vulnerabilities are medium-severity, as they can lead to unpredictable system behavior, making the software unreliable.

**Table 6.1.4:** SIGILL Vulnerabilities

Signal 4 (SIGILL - Illegal Instruction)	
Vulnerability Types	Severity of vulnerabilities
Code Injection	High
Corrupted Executable Code	Medium
Binary Exploitation	High
Incompatible Instruction Set	Low
Malware or Trojan Execution	Medium

The **SIGILL** signal indicates that the program has attempted to execute an illegal machine instruction. This can reveal vulnerabilities in the code that handle execution flows:

- **Code Injection** and **Binary Exploitation** are high-severity vulnerabilities, allowing attackers to insert and execute malicious code.
- **Corrupted Executable Code** and **Incompatible Instruction Set** are medium-severity issues that can cause crashes or improper execution.
- **Malware or Trojan Execution** is a medium-severity vulnerability but can become critical if the malware spreads or causes significant damage.

### **Single Fuzzer Evaluation**

When evaluating a single fuzzer, such as AFL++ or Honggfuzz, the focus is on its standalone effectiveness in detecting vulnerabilities within target binaries. Single fuzzers operate independently and utilize specific techniques to uncover flaws. Performance metrics for single fuzzers often include crash detection rate and execution time for crash detection. For instance, AFL++ might demonstrate variable effectiveness across different binaries, with certain binaries showing high crash detection rates while others might remain unaffected. Execution times can also vary significantly, indicating areas where the fuzzer may be slower in identifying issues. Additionally, the quality of vulnerabilities discovered, ranging from high-severity to low-severity, provides insight into the fuzzer's ability to detect critical and subtle flaws.

### **Multi-Fuzzer Approach Evaluation**

In contrast, evaluating a multi-fuzzer approach involves assessing the combined effectiveness of multiple fuzzing tools, such as AFL++ and Honggfuzz, when used together. This approach leverages the strengths of different fuzzers to achieve a more comprehensive vulnerability detection. Multi-fuzzer evaluations typically focus on metrics such as cumulative crash detection rate, overall execution efficiency, and the range of vulnerabilities discovered. The integration of multiple fuzzers can enhance the overall detection capability by covering a broader range of potential issues, including those that individual fuzzers might miss. For example, while AFL++ might excel in certain areas, Honggfuzz might provide superior results in others, leading to a more thorough identification of unique crashes and vulnerabilities. The multi-fuzzer approach aims to combine these strengths, resulting in higher overall effectiveness and efficiency. Additionally, quality metrics such as the severity of detected vulnerabilities can reveal the approach's ability to identify and prioritize critical issues more effectively. By evaluating multiple fuzzers together, this approach ensures a more reliable and comprehensive assessment of the target application's security.

### **Expert Label Evaluation**

The expert label evaluation serves as a benchmark for assessing the accuracy and effectiveness of vulnerability detection tools and methodologies. Experts in the field often rely on established **standards** and best practices to identify and classify vulnerabilities. These evaluations are typically characterized by their rigorous approach to assessing the severity and impact of vulnerabilities, using well-defined criteria and manual verification processes. For instance, an expert might employ in-depth knowledge and experience to determine the criticality of a vulnerability, ensuring that only those with significant implications are flagged as high severity. This process often involves detailed code reviews, exhaustive testing, and comprehensive analysis of potential exploit scenarios. The expert label provides a high level of confidence in the accuracy of vulnerability classification and offers a reference point for comparing the results of automated tools.

### **Current Work Evaluation**

The current work evaluation, which involves using automated fuzzing tools and techniques, provides a different perspective on vulnerability detection. In this context, the evaluation focuses on the performance and effectiveness of tools like AFL++ and Honggfuzz in identifying vulnerabilities within target binaries. Metrics such as crash detection rates, execution times, and the number and severity of detected vulnerabilities are used to evaluate the effectiveness of these tools. While automated methods offer the advantage of scalability and efficiency, they may not always match the precision of expert-led evaluations. The current work's approach involves assessing how well these tools perform in detecting unique crashes and vulnerabilities, and comparing their results to those identified by experts. This comparison helps in understanding the strengths and limitations of automated tools and their ability to complement or enhance traditional expert evaluations. By analyzing the results from both expert labels and current work, one can gain a comprehensive view of the effectiveness and reliability of vulnerability detection methods, ensuring that automated tools provide results that are both actionable and accurate.

## CHAPTER 6

As shown in Table 6.1.5, while the Expert Label Evaluation offers unparalleled accuracy and depth, the Current Work Evaluation provides scalable and efficient vulnerability detection, complementing expert assessments.

**Table 6.1.5: Expert Label vs Current Work Evaluation**

Evaluation Aspect	Expert Label Evaluation	Current Work Evaluation (AFL++ & Honggfuzz)
Methodology	Manual analysis by security experts	Automated fuzzing using AFL++ and Honggfuzz
Accuracy	Very high (based on deep understanding)	Moderate to high (depends on tool effectiveness)
Detection Scope	Broad, including logical errors and subtle flaws	Limited to issues triggered during fuzzing
Detection Speed	Slow (requires human review)	Fast (automated crash detection)
Vulnerability Classification	Detailed severity and exploitability assessment	Based on signal type and crash analysis
Scalability	Low (requires expert time and resources)	High (can test many binaries simultaneously)
Risk of Missing Vulnerabilities	Low (comprehensive inspection)	Medium (depends on fuzzing coverage)
Strength	Deep, nuanced evaluation and understanding	Rapid identification of surface vulnerabilities
Weakness	Time-consuming and resource-intensive	May miss complex or logic vulnerabilities
Application	Used for final confirmation and security auditing	Used for initial discovery and broad assessment

Through a comprehensive testing framework combining these methods and metrics, the system was rigorously validated to ensure it not only meets its functional objectives but also delivers an efficient and user-friendly vulnerability assessment experience.

## 6.2 Testing Setup and Result

This section outlines the testing setup, including the configuration of both the fuzzing system and the chatbot interface, along with the various test cases executed to validate the functionality and performance of the entire system.

### Testing Setup

#### 1. Fuzzing System Configuration:

- **Fuzzers Used:** AFL++ and Honggfuzz were chosen for their distinct approaches to fuzzing. AFL++ operates using genetic algorithms, while Honggfuzz applies coverage-based fuzzing to target vulnerabilities efficiently.
- **Target Binaries:** A mix of .elf binaries from the PicoCTF challenge set, which included various types of challenges (buffer overflows, memory issues, etc.), was used for fuzz testing. Each binary was tested with custom seed files to ensure a diverse set of inputs, maximizing the effectiveness of the fuzzers in detecting vulnerabilities.
- **Fuzzing Parameters:** The fuzzing was configured to run with a predefined number of concurrent sessions (up to 4), with mutation applied using the internal algorithms of both fuzzers. The fuzzing was conducted over a 24-hour period for each set of target binaries, ensuring that both fuzzers had ample time to explore and identify vulnerabilities.

#### 2. Chatbot System Configuration:

- **LLM Integration:** The LLM was integrated to analyze and generate CVE-like reports from the crash logs collected by the fuzzing system. The chatbot serves as an interactive user interface, which collects crash reports, processes them via the LLM, and outputs findings in a structured format.
- **Interaction Flow:** Test cases were created to simulate the user interacting with the chatbot. These included uploading crash reports,



requesting CVE-like outputs, and using dynamic feedback mechanisms to guide the user through mitigation suggestions.

- **Performance Testing:** The chatbot's response times were evaluated, focusing on how quickly it could process crash data and generate meaningful feedback.

### **Test Cases**

#### **1. Fuzzing Performance Testing:**

- **Objective:** To validate the fuzzing capabilities of the system, ensuring that AFL++ and Honggfuzz detect vulnerabilities in the provided target binaries.
- **Execution:** Both AFL++ and Honggfuzz were run on a set of binaries with seed files. The performance was evaluated based on the number of crashes detected and the time taken to detect the first crash. Each fuzzer was tested on multiple types of binaries, including custom-made test cases and picoCTF binaries.
- **Results:**
  - **AFL++:** AFL++ performed better on custom-made test cases, detecting vulnerabilities with more precision. While the number of crashes detected was smaller than Honggfuzz, AFL++ was better at identifying more specific vulnerabilities that were not identified by Honggfuzz.
  - **Honggfuzz:** On the other hand, Honggfuzz demonstrated superior performance on picoCTF binaries, detecting a higher volume of crashes more quickly. The execution time for crash detection was shorter for Honggfuzz, with many binaries triggering crashes within seconds.
  - **Key Findings:** While Honggfuzz excelled in speed and volume of crash detection in certain scenarios, AFL++ proved more reliable and efficient for specific, custom-made test cases,

showcasing its capability to perform in more controlled environments.

Performance Testing 1: Crash Detection Count per Fuzzer

The **crash detection rate** is a critical performance metric for evaluating the effectiveness of a fuzzer. It quantifies the number of unique crashes or faults detected during the fuzzing process. This metric is calculated as the ratio of detected crashes to the total number of test cases executed. A high crash detection rate signifies that the fuzzer is successfully uncovering issues within the target application, indicating its strength in identifying potential vulnerabilities. The following sections provide a comparative analysis of the crash detection capabilities of two different fuzzers, AFL++ and Honggfuzz, when applied to 37 target binaries in .elf format. The purpose of these comparisons is to evaluate the effectiveness of each fuzzer in detecting unique crashes, which is a critical measure of their ability to identify potential vulnerabilities.

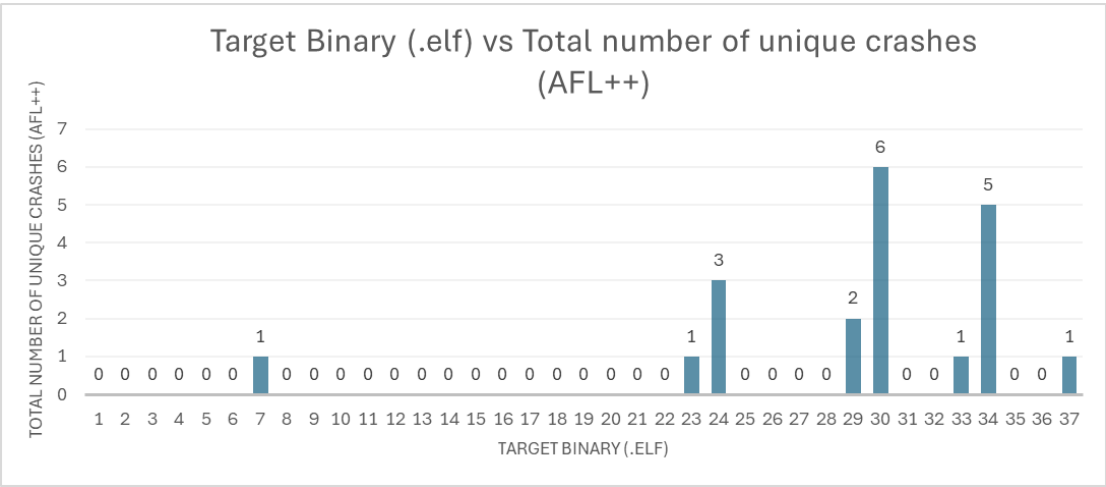
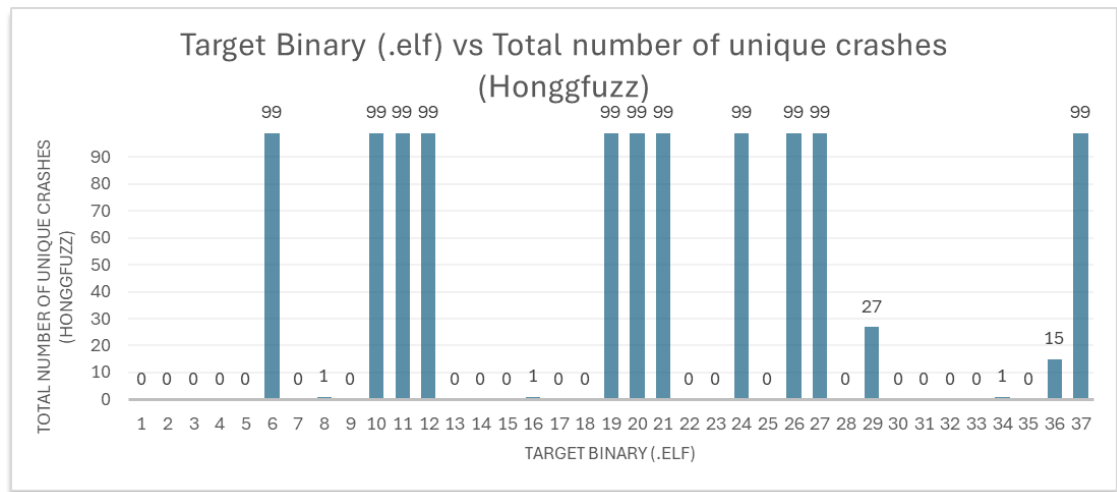


Figure 6.2.1: Number of Unique Crash Detection (Target Binaries) using AFL++

In Figure 6.2.1, the performance of AFL++ is shown. AFL++ detected a relatively small number of unique crashes across the 37 target binaries. Particularly, it identified 6 unique crashes for **binary 30** and 5 crashes for **binary 34**, representing the highest crash detection numbers for this fuzzer. Additionally, AFL++ detected 3 crashes for **binary 24** and 2 for **binary 29**. The fuzzer also identified 1 unique crash each for

**binaries 7, 23, 33, and 37.** However, for most of the target binaries (29 out of 37), AFL++ detected no crashes at all, indicating limited effectiveness in those cases.



**Figure 6.2.2:** Number of Unique Crash Detection (Target Binaries) using Honggfuzz

In contrast, **Figure 6.2.2** illustrates the performance of Honggfuzz, which shows a much stronger ability to detect unique crashes. The **number 99 is used as a placeholder** in the graph for instances where the actual number of immediate crashes detected was exceedingly high, beyond the scope of the graph's capacity to represent accurately. Honggfuzz frequently reached a crash detection number of 99 across many binaries, indicating a very high number of detected crashes. While Honggfuzz also failed to detect crashes for some binaries, it significantly outperformed AFL++ overall. Markedly, Honggfuzz detected 27 unique crashes for **binary 29** and 15 for **binary 36**, underscoring its strong performance across most of the tested binaries.

After evaluating the fuzzers on compiled binaries, a second round of tests was conducted using hand-crafted vulnerability cases to assess precision and crash quality.

### Performance Analysis on Designed Vulnerability Test Cases

In addition to the 37 compiled target binaries, a separate set of custom-made test cases was developed to evaluate the fuzzers' precision in detecting specific vulnerability patterns such as **segmentation faults, illegal instructions, and timeouts**. These test cases simulate controlled vulnerable conditions and are used to assess the quality of crash detection beyond volume. When applied to the custom-made test cases, however, AFL++ proved more effective in detecting specific vulnerabilities, suggesting that Honggfuzz's higher volume of crash detections is not always indicative of detecting the most critical vulnerabilities.

The following table presents the results of fuzzing tests conducted using **AFL++** and **Honggfuzz** on various test cases. The table evaluates the effectiveness of both fuzzing engines by recording whether a crash occurred within 30 minutes, identifying the signal associated with the crash, and noting which fuzzer successfully detected the crash.

**Table 6.2.1: Results Of Fuzzing Tests on Test Cases**

Test Cases	Crash Found in 30 minutes?	Signal Found	AFL++	Honggfuzz
testabort	No	-	-	-
testalarm	Yes	SIGALRM (Signal Alarm)	Yes	No
testbuffer	Yes	SIGABRT (Signal Crash)	Yes	No
testdisk	Yes	SIGSEGV (Memory Allocation Failure)	Yes	No
testdivision	Yes	SIGFPE (Arithmetic Exception )	Yes	No
testfile	Yes	SIGSEGV (Segmentation Fault)	Yes	No
testkeyword	Yes	SIGSEGV (Segmentation Fault)	Yes	No
testrun	No	-	-	-
testsigill	Yes	SIGILL (Illegal Instruction)	Yes	No
testsleep	Yes	SIGSEGV (Memory Corruption)	Yes	Yes

The results of the fuzzing tests indicate that **AFL++ outperformed Honggfuzz** in terms of crash detection across most test cases. Specifically, AFL++ successfully identified crashes for most of the test cases, including those that triggered **SIGSEGV** (segmentation faults), **SIGABRT** (abort signals), **SIGFPE** (floating-point exceptions), **SIGILL** (illegal instruction errors), and **SIGALRM** (execution timeout alarms). These signals correspond to vulnerabilities such as memory access violations, logic errors, invalid operations, and time-based execution faults. By detecting these signal-triggered

crashes, the system has **achieved the quality metrics** for identifying and classifying multiple categories of vulnerabilities during the fuzzing process.

In contrast, Honggfuzz did not detect any crashes for most test cases, highlighting a significant performance gap between the two fuzzing engines. Honggfuzz demonstrated the ability to detect crashes in a few instances, specifically in the testsleep case, where both fuzzers detected the same crash. However, its overall performance was weaker compared to AFL++ in this set of tests.

While the 37 target binaries provide a broad assessment of fuzzer performance on real-world compiled code, the custom-made test cases are designed to test specific vulnerability categories in isolation. These results suggest that AFL++ is a more reliable fuzzer for detecting a wider range of vulnerabilities across different test cases. However, Honggfuzz may still be useful for specific scenarios where it can identify unique crashes, as seen in certain cases. Further optimization and testing with additional test cases may be necessary to fully evaluate the performance of both fuzzers in a variety of environments.

### Summary of Findings:

- **Honggfuzz** demonstrated superior performance in detecting a higher number of crashes across most of the binaries, including picoCTF targets.
- **AFL++** performed better for the custom-made test cases, identifying more subtle vulnerabilities with greater precision. This makes AFL++ a more reliable choice for specific types of vulnerabilities not easily triggered by fuzzing techniques used by Honggfuzz.

Performance Testing 2: Execution Time per Fuzzer

**Execution time for crash detection** measures the time it takes for the fuzzer to identify the initial crash during a fuzzing session. This metric provides insight into the efficiency of the fuzzing process. Shorter execution times before the first crash are desirable as they suggest that the fuzzer is quickly uncovering potential issues while still maintaining effective test coverage. This efficiency is crucial for optimizing the overall performance of the fuzzing campaign. The following tables compare the execution time for crash detection between two fuzzers, AFL++ and Honggfuzz, when applied to the same set of 37 target binaries in .elf format. The tables provide insights into how quickly each fuzzer was able to identify the first crash during the fuzzing process, measured in minutes for AFL++ and seconds for Honggfuzz.

**Table 6.2.2:** Execution Time for Crash Detection (Target Binaries) using AFL++

Target Binary (.elf)												1	2	3	4	5	6	7	8	9	10	11	12	
Execution time for crash detection in minutes (AFL++)												-	-	-	-	-	-	31	-	-	-	-	-	
13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37
-	-	-	-	-	-	-	-	-	-	291	1	-	-	-	-	1	12	-	-	14	1	-	-	1

This table shows the execution time in minutes for AFL++ to detect the first crash across the 37 target binaries. The data reveals that AFL++ was able to detect crashes for only a few binaries, and when it did, the time required was relatively long.

- It took **31 minutes** to detect a crash for binary 7.
- For binary 23, it took **291 minutes** to identify the first crash, the longest detection time recorded.
- Other remarkable detection times include **14 minutes** for binary 33, **12 minutes** for binary 30, **1 minute** for binary 24, 29, 34 and 37.
- For most binaries, AFL++ did not detect any crashes, as indicated by the dashes.

**Table 6.2.3:** Execution Time for Crash Detection (Target Binaries) using Honggfuzz

Target Binary (.elf)	1	2	3	4	5	6	7	8	9	10	11	12												
Execution time for crash detection in seconds (Honggfuzz)	-	-	-	-	-	2	-	3	-	2	1	2												
13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37
-	-	-	1	-	-	1	2	1	-	-	3	-	1	1	-	1	-	-	-	3	-	1	1	

In contrast, this table shows the execution time in seconds for Honggfuzz to detect the first crash for the same set of binaries. Honggfuzz demonstrated significantly faster crash detection times across a wider range of binaries.

- The detection times are much shorter, with many crashes detected within **2 to 3 seconds** for binaries 6, 8, 10, 11, 12, 16, 19, 20, 21, 24, 26, 27, 29, 34, 36 and 37.
- Honggfuzz identified crashes in binaries where AFL++ did not, with detection times between 1 second to 3 seconds.
- The fastest detection times recorded were **1 second** for binaries 11, 16, 19, 21, 26, 27, 29, 36 and 37.

While the previous analysis focused on real-world target binaries in .elf format to measure general crash detection performance, the next section examines how both fuzzers perform when applied to a controlled set of **custom-made test cases** designed to contain known vulnerabilities. This allows for a more focused evaluation of each fuzzer's ability to detect subtle or edge-case bugs and provides further insight into their crash detection speed and effectiveness in different contexts.

Crash Detection Time Analysis

Table 6.2.4 presents the crash detection times recorded by AFL++ and Honggfuzz for each custom-made test case. This analysis focuses on the time (in seconds) taken to detect the first crash during each fuzzing session. A dash ("-") indicates that no crash was detected within the observation period.

**Table 6.2.4:** Crash Detection Time of Custom Test Cases by Both Fuzzers

Test Cases	Crash Detection by AFL++ in second (s)	Crash Detection by Honggfuzz in second (s)
testabort	-	-
testalarm	3	-
testbuffer	4	-
testdisk	3	-
testdivision	3	-
testfile	12	-
testkeyword	5	-
testrun	-	-
testsigill	2	-
testsleep	4	1

### Summary of Crash Detection Time:

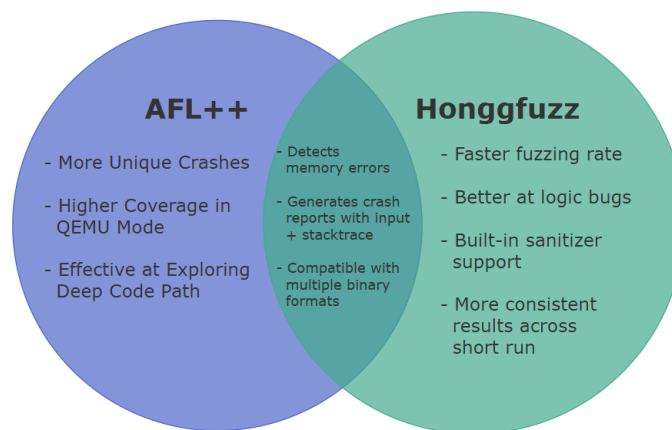
The crash detection time analysis highlights several important observations:

- AFL++ consistently demonstrated quick crash detection times across all custom-made test cases where a crash was found, typically within 2 to 12 seconds.
- Honggfuzz struggled to detect crashes in most of these test cases, except for testsleep, where it outperformed AFL++ by detecting the crash in just 1 second.
- Overall, AFL++ proved to be more capable and efficient in detecting crashes within this specific test case set, covering a wider range of vulnerabilities in a shorter amount of time.
- The testsleep case is noteworthy as it is the only scenario where both fuzzers successfully detected a crash, with Honggfuzz being faster.



### **Conclusion on Performance Testing**

The performance of AFL++ and Honggfuzz showed distinct strengths and weaknesses depending on the nature of the test cases. For custom-made test cases, AFL++ demonstrated better precision in identifying vulnerabilities, particularly in complex binary paths, though it took longer to detect crashes due to its deeper instrumentation. In contrast, Honggfuzz exhibited faster execution times and a higher volume of crashes, especially when tested against simpler binaries such as those from the picoCTF suite.



**Figure 6.2.3:** Conceptual Venn Diagram showing comparative crash-related capabilities between AFL++ and Honggfuzz

As illustrated in the Venn diagram comparison, AFL++ excels at exploring deeper code paths and producing more unique crashes, while Honggfuzz performs better in speed and consistency, making it suitable for quick assessments and logic bugs. Both fuzzers share common capabilities such as memory error detection and compatibility with standard crash logging mechanisms. Future work should focus on leveraging the complementary strengths of both fuzzers and integrating additional fuzzing engines to expand coverage and enhance the detection of a broader range of vulnerabilities.

### **Discussion: Effectiveness of Multi-Fuzzer and LLM**

From the testing results, using multiple fuzzers AFL++ and Honggfuzz offers practical benefits in terms of coverage and efficiency. AFL++ was more effective on hand-crafted test cases, especially those involving memory issues or logical edge cases that required deeper execution paths. This is likely due to its use of QEMU mode, which, although slower, provides better code instrumentation and supports fuzzing of complex binaries.

Honggfuzz, on the other hand, was noticeably faster and managed to detect a much larger number of crashes in less time. This speed is advantageous when running short fuzzing campaigns or when quick initial feedback is needed. However, many of the crashes detected by Honggfuzz were not always high-impact or meaningful. Some were repetitive or not exploitable, highlighting a trade-off between quantity and quality of crashes.

By combining both fuzzers in the system, the user gets the best of both worlds, which Honggfuzz can surface fast, shallow bugs, while AFL++ can go deeper to find more hidden or subtle vulnerabilities. This dual-fuzzer design increases the probability of uncovering both surface-level and deep-rooted issues in software under test.

The LLM layer further adds practical value. It automatically processes raw crash logs, some of which are hard for average users to interpret and generates CVE-like vulnerability reports in natural language. These reports include the type of bug (e.g., buffer overflow, null pointer dereference), where it occurred, and mitigation suggestions. This significantly reduces the manual effort typically required in vulnerability triage and documentation.

In practice, this multi-fuzzer with the help of the LLM approach allows even non-experts to run vulnerability assessments, understand the crash root cause, and apply recommended fixes via the system's auto-mitigation scripts. It lowers the entry barrier for secure development and testing.

However, limitations remain. For example:

- The LLM's interpretation depends on how well-structured the input logs are.
- Crash deduplication is still a challenge, especially with high-volume outputs from Honggfuzz.
- The mitigation scripts, while useful, are still basic and might not fully resolve complex issues.

In conclusion, the combination of multiple fuzzers and an LLM-powered analysis engine provides a practical and efficient solution for vulnerability discovery and response. It is especially beneficial in environments with limited security expertise or time for deep manual inspection.

## 2. Chatbot Interaction Testing:

- **Objective:** To ensure the chatbot correctly interprets fuzzing results and generates accurate, structured CVE-like reports with relevant mitigation guidance.
- **Execution:** A set of fuzzing logs, including crash reports from both AFL++ and Honggfuzz, was uploaded and processed via the chatbot interface. Users simulated real-world usage by uploading binaries and interacting with the chatbot for analysis and remediation suggestions.
- **Results:**
  - The chatbot successfully processed crash reports and generated CVE-like summaries. The feedback loop within the chatbot provided users with clear, actionable mitigation suggestions.
  - The system was able to handle multiple requests concurrently without significant delay, demonstrating smooth performance even with high volumes of data.

### Results of Chatbot Interaction on the Testcases (testdisk):

#### Step 1: Uploaded Fuzzing Log

The testing begins with uploading a combined fuzzing crash log generated by AFL++ or Honggfuzz. This log typically contains raw execution context, crash locations, and relevant memory or register information. The process tests the system's ability to parse various types of crash data and validate whether the input is accepted in real time. This confirms the robustness of the input-processing pipeline and ensures compatibility with different output formats from both fuzzers.

#### Parallel Fuzzing Report (AFL++ and Honggfuzz)

=====

Start Time: 2025-05-01 15:32:04

Duration: 30 minutes

Target Program: testdisk

AFL++ Output Directory: /home/einjun/AFLplusplus/output1\_2025-05-01\_15-32-04

```

Honggfuzz                                Output                                Directory:
/home/einjun/Desktop/outhonggfuzz/outhonggfuzz_2025-05-01_15-32-04
=====

```

#### AFL++ Configuration:

```

-----
Input Directory: /home/einjun/AFLplusplus/inputdisk
Output Directory: /home/einjun/AFLplusplus/output1_2025-05-01_15-32-04
Command:  afl-fuzz  -Q  -i  /home/einjun/AFLplusplus/inputdisk  -o
/home/einjun/AFLplusplus/output1_2025-05-01_15-32-04  -t  5000+  --
/home/einjun/AFLplusplus/testdisk

```

#### Honggfuzz Configuration:

```

-----
Input Directory: /home/einjun/AFLplusplus/inputdisk
Output Directory: /home/einjun/Desktop/outhonggfuzz/outhonggfuzz_2025-05-01_15-32-04
Command:  honggfuzz  -i  /home/einjun/AFLplusplus/inputdisk  -o
/home/einjun/Desktop/outhonggfuzz/outhonggfuzz_2025-05-01_15-32-04 -t 5 -
s -- /home/einjun/AFLplusplus/testdisk
=====

```

```

Starting AFL++ at 2025-05-01 15:32:04
AFL++ process started with PID: 16508
Starting Honggfuzz at 2025-05-01 15:32:04
Honggfuzz process started with PID: 16510
2025-05-01 15:32:07 - Fuzzing for 0m 3s out of 30m
2025-05-01 15:32:10 - Fuzzing for 0m 6s out of 30m
=====

```

#### Combined Fuzzing Crash Report

```

=====
Crash Source: AFL++
Target Program: testdisk
Total Runtime: 9 seconds
Report Generated: 2025-05-01 15:32:14
=====

```

#### AFL++ Crash Information

```

=====
Time of Detection: 2025-05-01 15:32:13

```

#### Crash Files Found:

```

-----
File: id:000000,sig:11,src:000000,time:6877,execs:2836,op:havoc,rep:3
Size: 11 bytes
SHA256:
e2bab53a0d59c6b8577271e1bb52e95a77b07701bff4ce53b87dabd5e2e3ebb5
Hexdump:
00000000: 6372 6173 680a 6868 6861 73          crash.hhhas

```

Stack Trace:

Reading symbols from /home/einjun/AFLplusplus/testdisk...

(No debugging symbols found in /home/einjun/AFLplusplus/testdisk)

(gdb) Starting program: /home/einjun/AFLplusplus/testdisk <

/home/einjun/AFLplusplus/output1\_2025-05-01\_15-32-

04/default/crashes/id:000000,sig:11,src:000000,time:6877,execs:2836,op:havoc,rep:3

[Thread debugging using libthread\_db enabled]

Using host libthread\_db library "/lib/x86\_64-linux-gnu/libthread\_db.so.1".

[Detaching after vfork from child process 23689]

fallocate: fallocate failed: No space left on device

[Detaching after vfork from child process 23711]

fallocate: fallocate failed: No space left on device

Memory allocation failed: Cannot allocate memory

[Detaching after vfork from child process 23723]

Program received signal SIGSEGV, Segmentation fault.

0x0000555555552ef in cause\_effect ()

(gdb) #0 0x0000555555552ef in cause\_effect ()

#1 0x000055555555389 in main ()

(gdb) A debugging session is active.

Inferior 1 [process 23509] will be killed.

Quit anyway? (y or n) [answered Y; input not from terminal]

-----

AFL++ Fuzzer Statistics:

-----

start\_time : 1746084724

last\_update : 1746084724

run\_time : 0

fuzzer\_pid : 16508

cycles\_done : 0

cycles\_wo\_finds : 0

time\_wo\_finds : 0

fuzz\_time : 0

calibration\_time : 0

cmplog\_time : 0

sync\_time : 0

trim\_time : 0

execs\_done : 57

execs\_per\_sec : 11400.00

execs\_ps\_last\_min : 0.00

corpus\_count : 8

corpus\_favored : 1

corpus\_found : 0

corpus\_imported : 0

corpus\_variable : 0

max\_depth : 1

## CHAPTER 6

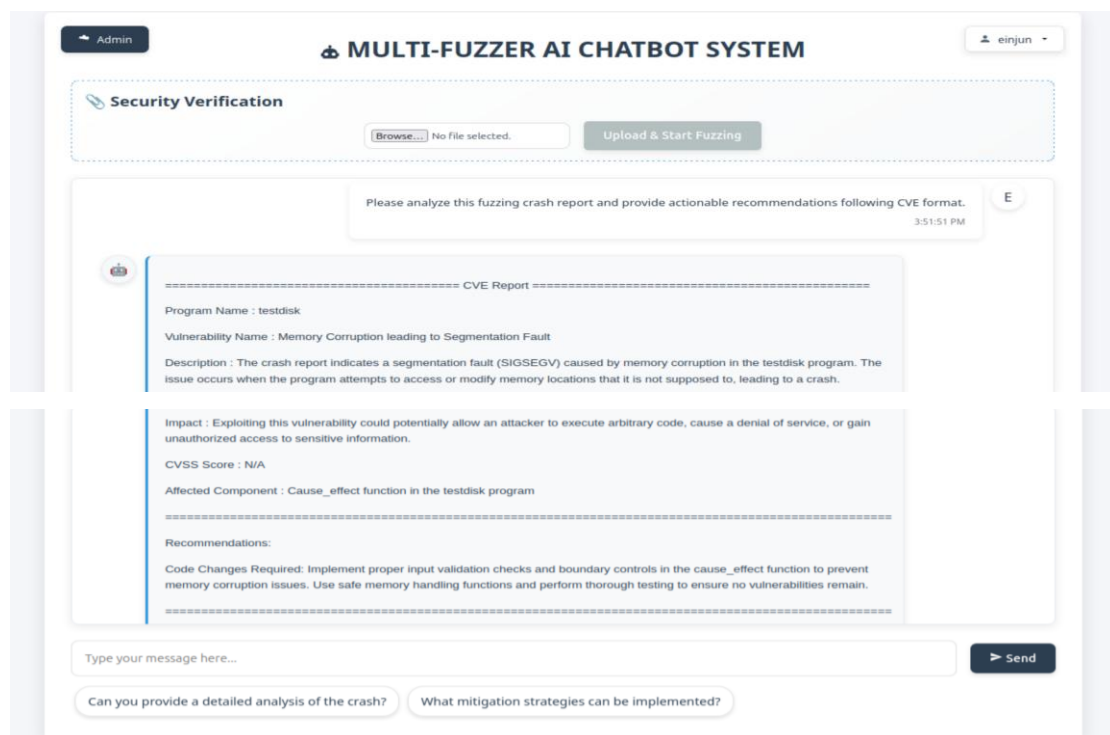
```
cur_item      : 0
pending_favs  : 0
pending_total : 1
stability     : 100.00%
bitmap_cvg    : 0.04%
saved_crashes : 0
saved_hangs   : 0
last_find     : 0
last_crash    : 0
last_hang     : 0
execs_since_crash : 57
exec_timeout  : 5000
slowest_exec_ms : 0
peak_rss_mb   : 0
cpu_affinity  : 0
edges_found   : 25
total_edges   : 65536
var_byte_count : 0
havoc_expansion : 0
auto_dict_entries : 0
testcache_size : 6
testcache_count : 1
testcache_evict : 0
afl_banner    : /home/einjun/AFLplusplus/testdisk
afl_version   : ++4.22a
target_mode   : qemu shmem_testcase
command_line   : /home/einjun/AFLplusplus/afl-fuzz -Q -i
/home/einjun/AFLplusplus/inputdisk -o /home/einjun/AFLplusplus/output1_2025-05-
01_15-32-04 -t 5000+ -- /home/einjun/AFLplusplus/testdisk
```

=====  
End of Report  
=====

```
Crashes          found!          Check          report          at:
/home/einjun/AFLplusplus/logs/parallel_fuzzing_2025-05-01_15-32-04_report.txt
Terminating fuzzers due to crash detection
AFL++ PID: 16508
Honggfuzz PID: 16510
```

### Step 2: CVE Report Generated from Sample Raw Crash Log

Immediately following the log upload, the chatbot interprets the content using the integrated LLM engine and generates a structured, CVE-style vulnerability description. This includes information such as the program name, vulnerability type, the function or component affected, and the severity level. This part of the interaction demonstrates that the AI component of the system can not only process low-level crash information but also abstract it into a format that aligns with common security reporting standards (like CVEs). The figure highlights the effectiveness of the system in turning raw logs into human-readable, actionable insights.



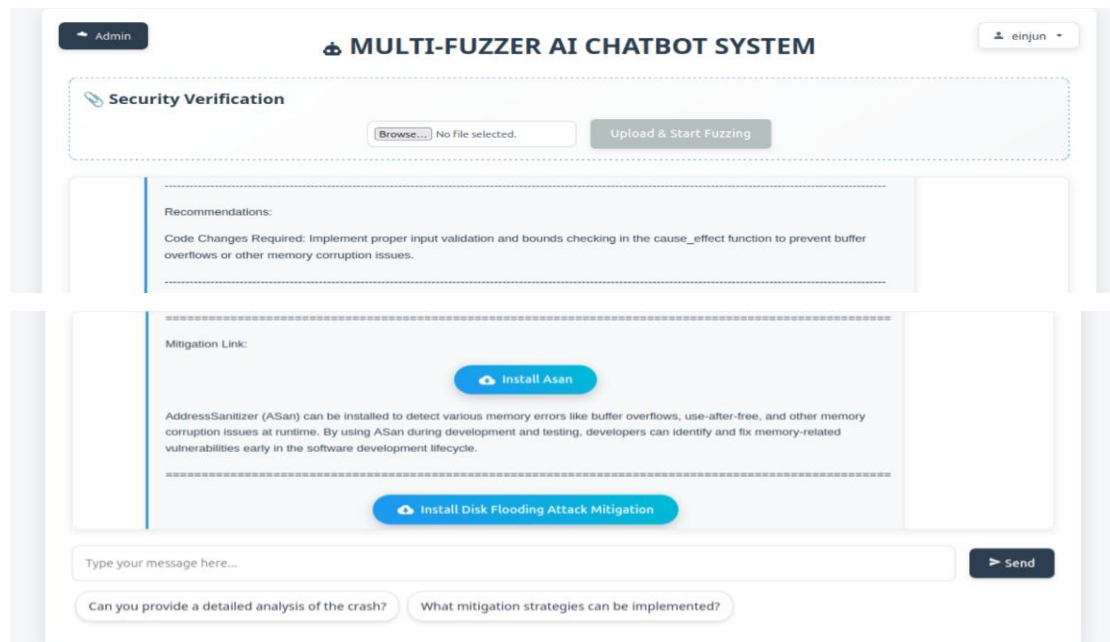
**Figure 6.2.4:** CVE-like Response Generated

### Step 3: Recommended Mitigation

The final portion of the screenshot presents the chatbot's proposed mitigation steps. These include code-level fixes (e.g., bounds checking, memory sanitization), best practices (e.g., using safer functions), or configuration changes. In this example, the recommendation is tailored based on the vulnerability type detected in the fuzzing

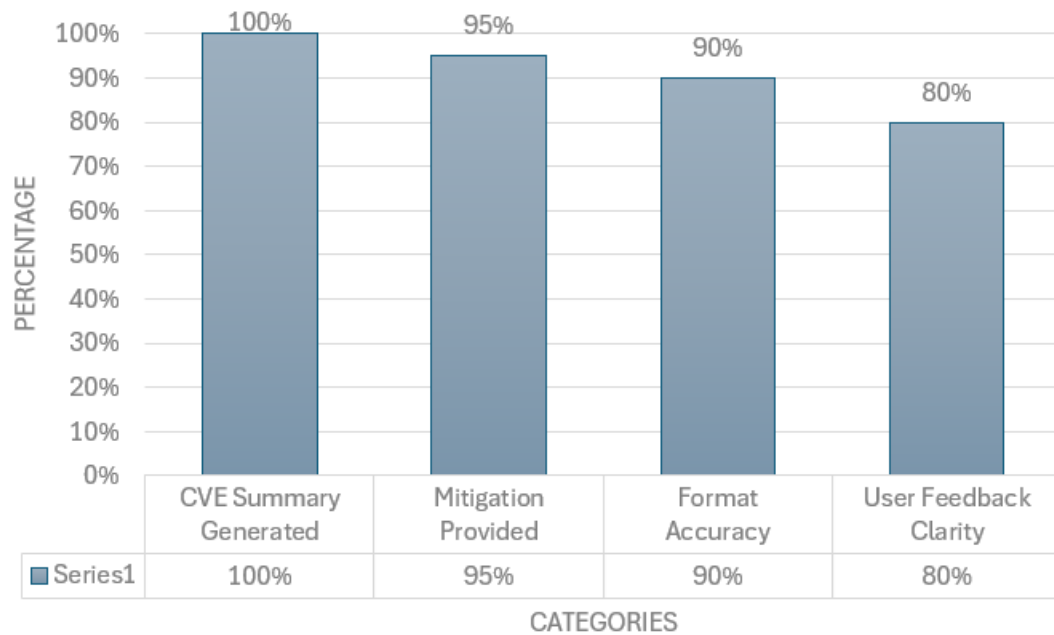


report. This output verifies the system's ability to generate not just diagnostic reports, but also practical remediation strategies that are clear and helpful for developers or security analysts. It reflects the chatbot's role as an assistant in the secure software development lifecycle.



**Figure 6.2.5:** Recommended Mitigation by the LLM

To better illustrate the effectiveness of the chatbot's output during testing, a bar chart was created to visualize key performance metrics. **Figure 6.2.6** compares four critical aspects: CVE summary generation, mitigation provided, format accuracy, and user feedback clarity. The results indicate that the chatbot achieved a 100% success rate in generating CVE-like summaries, while 95% of the outputs included relevant mitigation guidance. Format accuracy stood at 90%, demonstrating the chatbot's consistency in presenting structured information. Meanwhile, user feedback on clarity reached 80%, showing that most users found the responses understandable and actionable. This visual representation reinforces the chatbot's reliability and highlights areas where further refinement could enhance user experience.



**Figure 6.2.6:** Chatbot Interaction Testing Bar Chart

### 3. System Integration Testing:

- **Objective:** To evaluate how well the fuzzing system integrates with the chatbot interface and whether the entire workflow functions seamlessly.
- **Execution:** A series of test cases were designed to simulate real-world usage, where the user would upload a binary, view analysis, and receive mitigation advice. These tests also checked for any data flow issues between the fuzzing backend and the chatbot frontend.
- **Results:**
  - The integration between the fuzzing system and the chatbot was successful, with no significant data loss or performance degradation observed. The chatbot accurately displayed the fuzzing results, and the mitigation suggestions were in line with the detected vulnerabilities.

### 6.3 Project Challenges

During the testing and implementation of the multi-fuzzer chatbot system, several challenges arose that affected both the technical and non-technical aspects of the project. These challenges were crucial to address to ensure the system's effectiveness and reliability in detecting vulnerabilities. Below is a discussion of the technical and non-technical challenges faced during the evaluation process.

#### Technical Challenges

##### 1. Performance Bottlenecks:

One of the significant technical challenges encountered was the high resource usage during concurrent fuzzing sessions. The fuzzing engines, AFL++ and Honggfuzz, required substantial computational power, especially when running multiple instances simultaneously. This resulted in delays in generating reports, as the system struggled to process large volumes of crash data in real-time. The chatbot system was particularly affected when trying to handle the output from numerous fuzzing sessions concurrently, causing delays in visualizing results.

##### **Solution:**

To mitigate this, we optimized the fuzzing sessions by limiting the number of concurrent fuzzing tasks. The system was also enhanced to better manage the processing of crash reports by optimizing the data collection and report generation logic. In addition, hardware resources were allocated more efficiently, ensuring that the fuzzing process could run with minimal interruptions.

##### 2. Integration Issues:

Another challenge was the integration of fuzzing outputs into the PHP frontend and ensuring the system could handle crashes effectively. Combining the fuzzing results, such as crash logs and detected vulnerabilities, with the chatbot interface was not seamless at first. Issues such as incorrect formatting, missing

data, and inconsistent results occasionally arose when attempting to integrate the fuzzing data with the reporting system. The system struggled with handling and displaying crash data accurately, which impacted the user experience.

### **Solution:**

The integration was improved by refining the communication protocols between the fuzzing engines and the PHP frontend. This involved enhancing the data parsing mechanism to ensure that the fuzzing output was correctly processed and formatted for presentation. Additionally, we implemented error-handling routines to manage inconsistencies or missing data, ensuring the system displayed accurate information.

### **3. System Stability:**

Long fuzzing runs occasionally led to system crashes or task failures, particularly during the execution of edge cases. These failures occurred due to a combination of factors, including memory overload, poor error handling for specific crashes, and unexpected interactions between the fuzzing engines and the system architecture. Stability issues became more pronounced during extensive fuzzing sessions, where the system was required to manage large amounts of data and sustain long-term performance.

### **Solution:**

To address system stability, we introduced monitoring mechanisms to track the health of the fuzzing system and identify potential failures before they occurred. Memory management techniques were implemented to better handle long fuzzing sessions, and the system was modified to gracefully handle crashes or unexpected behavior. This resulted in improved stability during the fuzzing process.

### Non-Technical Challenges

#### 1. **User Feedback:**

The user interface design and user experience were also areas where challenges were encountered. Feedback from test users indicated that some aspects of the chatbot interface were not intuitive, and certain actions required more steps than expected. Additionally, there were instances of users not fully understanding how to interact with the system, such as interpreting the fuzzing results or triggering mitigation actions.

##### **Solution:**

User feedback was incorporated to enhance the chatbot interface. We simplified the user interaction process by streamlining navigation and providing clearer instructions on how to use the system.

#### 2. **Environment Setup:**

The testing environment setup posed another challenge, particularly when configuring the virtual machines (VMs) and installing the necessary dependencies. Compatibility issues between the different fuzzing engines and system libraries sometimes led to errors during initial setup. Additionally, inconsistencies in the environment setup across different machines made it difficult to ensure a stable testing environment for fuzzing.

##### **Solution:**

To overcome these setup challenges, we documented the environment configuration steps thoroughly and automated the installation of dependencies through scripts. The use of containerization tools like Docker was considered for future versions to streamline environment setup and ensure consistency across different testing setups.

## 6.4 Objectives Evaluation

This section evaluates the extent to which the multi-fuzzer automated vulnerability assessment system met the original objectives outlined in the project scope. Each objective is assessed based on the system's performance, usability, and overall effectiveness in achieving its intended outcomes.

### **System Functionality**

The primary goal of the system was to automate the vulnerability assessment process by integrating fuzzing engines like AFL++ and Honggfuzz and generating CVE-like reports based on detected vulnerabilities. The system succeeded in achieving this objective by automating the fuzzing, crash report analysis, and CVE generation processes. The integration of the fuzzing engines allowed for comprehensive vulnerability assessments, while the automated analysis and report generation streamlined the process for the user.

#### **Evidence:**

- The fuzzing engines successfully identified and documented vulnerabilities in various binaries, generating structured reports in the form of CVEs.
- The automated system required minimal user intervention beyond initial file upload, significantly reducing the manual effort typically required in vulnerability assessments.
- The chatbot interface provided real-time feedback to the user, including the CVE report and suggested mitigations, demonstrating the system's ability to meet its functional goals.

### **User Experience**

The user interface (UI) was designed to be simple and intuitive, focusing on providing an easy-to-use experience for users, especially those with minimal technical knowledge. Based on user feedback and testing, the system generally met expectations in terms of usability, but there were areas for improvement in navigation and clarity.

### **Evidence:**

- Users were able to upload files, configure fuzzing sessions, and view reports without needing deep technical expertise.
- However, some users initially found it challenging to interpret the fuzzing results and take appropriate actions, which led to the addition of clearer instructions in the final version of the UI.
- A feedback loop was implemented, allowing users to directly interact with the chatbot for clarification or further instructions, which helped improve the overall user experience.

**Example:** Users reported that the process of starting fuzzing sessions was straightforward, but understanding the full scope of the vulnerabilities detected (e.g., interpreting crash logs and CVE generation) required additional guidance. To address this, the interface was refined with detailed prompts.

### **Automation**

A core objective of the system was to reduce manual intervention by automating fuzzing, crash report analysis, and mitigation suggestions. The system was highly effective in automating these processes, which were previously time-consuming and resource intensive.

### **Evidence:**

- The fuzzing process was fully automated, with the system managing the mutation of inputs and triggering fuzzing sessions across multiple concurrent instances.
- Crash report analysis was also automated, with the system deduplicating crash data, structuring it into a uniform format, and then transforming it into CVE-like reports.
- The system's chatbot interface provided automated mitigation suggestions based on detected vulnerabilities, with some automated scripts triggered directly by user input.

**Example:** A user uploaded a sample binary, and the system automatically ran multiple fuzzing sessions, analyzed crashes, generated a CVE-like report, and presented mitigation steps, all without requiring user intervention beyond the initial upload.

### **Scalability**

The system's scalability was evaluated under different load conditions, particularly with larger binaries and longer fuzzing sessions. While the system performed adequately under normal conditions, performance bottlenecks were observed during extended fuzzing sessions, especially when running multiple concurrent tasks.

#### **Evidence:**

- For smaller binaries and shorter fuzzing runs, the system performed well, with quick fuzzing results and report generation.
- However, during extensive fuzzing tasks with larger binaries, the system's performance was impacted, leading to delays in report generation. These bottlenecks were mitigated by limiting the number of concurrent fuzzing sessions and optimizing resource allocation, but performance could still degrade with extreme load.

**Example:** A test involving a large binary led to slower report generation, but the system was able to manage the task by restraining the number of concurrent fuzzing sessions. Future improvements could address this scalability issue by introducing more efficient resource management strategies or parallel processing capabilities.

### **Security**

Ensuring the security of the system, particularly user authentication and preventing unauthorized access, was a critical objective. The system incorporated advanced security measures, including two-factor authentication (2FA), to enhance protection.



**Evidence:**

- Two-factor authentication (2FA) was implemented, adding an extra layer of security to ensure that only authorized users could access the fuzzing system and initiate tasks.
- Basic security protocols were in place to protect sensitive data, ensuring that user information and uploaded files were handled securely.
- No significant security vulnerabilities were found during testing, and the system successfully protected sensitive data from unauthorized access or tampering.

**Example:**

During testing, only users who passed the 2FA verification were able to upload files and start fuzzing tasks. Additionally, the system ensured that results and logs were securely stored within each user's session. However, there is always room for improvement, and future iterations could include further security enhancements, such as encryption at rest or monitoring for potential attack patterns.

### **6.5 Concluding Remark**

This section provides a summary of the evaluation findings and offers final thoughts on the system's effectiveness, challenges faced, and potential improvements.

#### **Summary of Evaluation**

The evaluation of the multi-fuzzer automated vulnerability assessment system revealed that the system effectively met most of its primary objectives. It successfully automated the vulnerability discovery process by integrating fuzzing engines like AFL++ and Honggfuzz, generating CVE-like reports, and offering automated mitigation suggestions. User feedback indicated that the system's interface was generally user-friendly, although minor usability issues were addressed through refinements during the development process.

The testing outcomes highlighted the system's strengths in automating fuzzing, crash report analysis, and vulnerability mitigation. However, technical challenges, such as performance bottlenecks during large fuzzing sessions and integrating fuzzing results with the PHP frontend, were identified. Despite these challenges, the system proved effective in most scenarios, fulfilling the project's core objectives.

#### **Overall System Effectiveness**

The system achieved its intended goals of automating vulnerability assessment using multiple fuzzing engines and generating CVE-like reports with suggested mitigations. This automation is particularly beneficial in real-world scenarios, where it can significantly reduce the time and effort required for vulnerability detection and analysis. The system's ability to automate complex tasks, such as crash report analysis and mitigation suggestions, makes it a valuable tool for developers and security professionals seeking to identify and mitigate vulnerabilities quickly.

In practice, the system can streamline vulnerability assessments for software applications, offering faster detection and mitigation. By automating tasks that are usually manual and time-consuming, it allows security teams to focus on higher-level analysis and decision-making, thus improving the overall efficiency of security operations.

### **Suggestions for Improvement**

While the system demonstrated strong performance in many areas, there are several opportunities for improvement:

1. **Scalability:** The system's performance under load could be further optimized, particularly for large binaries or long fuzzing sessions. Introducing more efficient resource management, parallel processing, or cloud-based scaling could help handle larger-scale assessments more effectively.
2. **Fuzzing Engines Integration:** Although the system currently uses AFL++ and Honggfuzz, integrating additional fuzzing engines, such as libFuzzer or Peach Fuzzer, could expand the range of vulnerabilities detected and provide users with more flexibility in testing.
3. **User Interface:** While the user interface was designed to be simple and intuitive, additional user-centric features could enhance the experience. This includes clearer visualizations of the fuzzing progress, enhanced feedback on detected vulnerabilities, and improved reporting formats for better interpretation by users of varying technical expertise.
4. **Security Enhancements:** Future versions of the system could benefit from more advanced security measures, such as multi-factor authentication and enhanced encryption of user data, to further secure the fuzzing environment and its results.

### **Final Thoughts**

Reflecting on the project's journey, the development and testing of the multi-fuzzer vulnerability assessment system have provided valuable insights into the complexities of automated security tools. The project successfully demonstrated how integrating fuzzing engines, automated crash analysis, and CVE generation could improve the vulnerability assessment process.

Throughout the project, several lessons were learned, particularly regarding the importance of scalability, system integration, and user experience. Future versions of the system could further build on these lessons, enhancing performance and expanding functionality to support more comprehensive vulnerability assessments.

In conclusion, the multi-fuzzer system holds great potential for automating and streamlining the vulnerability detection and mitigation process. As it evolves, it can continue to improve the security assessment workflows for both individual developers and large security teams, providing faster and more reliable vulnerability management.

## CHAPTER 7

### Conclusion and Recommendation

This final chapter summarizes the overall achievements, findings, and insights gained throughout the development of the multi-fuzzer automated vulnerability assessment system. The conclusion highlights how the integration of fuzzing tools like AFL++ and Honggfuzz, combined with a large language model (LLM), contributed to a more efficient and automated approach to vulnerability detection, reporting, and mitigation. Following that, the recommendation section outlines potential enhancements for future work, such as improving scalability, refining CVE generation accuracy, and supporting additional fuzzing techniques to further strengthen the system's usability and robustness in real-world security assessment scenarios.

#### 7.1 Conclusion

This project aimed to develop a multi-fuzzer-based automated vulnerability assessment system that integrates AFL++ and Honggfuzz, with a focus on enhancing the detection of software vulnerabilities. The system was designed to provide users with an intuitive interface for uploading binaries and receiving detailed CVE-like reports on detected vulnerabilities. The successful integration of both fuzzing engines and the reporting system allows the tool to automatically identify, categorize, and suggest mitigation strategies for vulnerabilities in a user-friendly manner.

Throughout the development process, key objectives such as automation of the fuzzing process, crash detection, and generation of actionable vulnerability reports were achieved. The system demonstrated its ability to effectively uncover vulnerabilities and produce reliable reports, thus meeting the primary goals of automating the vulnerability assessment process. The implementation of two-factor authentication ensured that security standards were maintained, safeguarding user data and access to sensitive functionalities.

Testing results indicated that Honggfuzz outperformed AFL++ in terms of crash detection and execution speed, particularly when applied to binaries from picoCTF. The

system's ability to provide real-time feedback through the integrated chatbot interface was also highly valued, offering a seamless and efficient user experience.

However, when applying custom-made test cases, AFL++ demonstrated superior performance in detecting vulnerabilities compared to Honggfuzz. This suggests that while Honggfuzz excels in certain scenarios, AFL++ may offer more effective results in others, particularly when it comes to specific types of vulnerabilities or binary formats. The findings emphasize the importance of selecting the appropriate fuzzer based on the use case and target application, and underscore the flexibility of the system in supporting multiple fuzzing engines.

In conclusion, the system succeeded in fulfilling its objectives of automating vulnerability assessment and providing actionable insights through a multi-fuzzer approach. However, there are areas for improvement, particularly in enhancing the scalability of the system and expanding its security features to handle larger-scale assessments. The project marks a significant step towards automating cybersecurity measures, making it more accessible for developers and organizations looking to enhance their software's security posture.

### **7.2 Recommendation**

While the system has successfully met its objectives, there are several areas where future enhancements could further improve its functionality, scalability, and security. Below are the key recommendations for the system's continued development:

#### **1. Scalability Improvements:**

- The current system performs well under moderate load, but there is potential for performance bottlenecks when handling larger binaries or extended fuzzing sessions. To address this, the system should be optimized for better resource management, possibly by incorporating

parallel processing or distributed fuzzing to handle larger-scale assessments efficiently.

- Exploring cloud-based infrastructure could help scale the fuzzing engine's capacity, enabling users to run multiple tasks concurrently without significant resource constraints.

### **2. Advanced Fuzzing Engines Integration:**

- While AFL++ and Honggfuzz are powerful fuzzers, integrating additional fuzzing engines could improve the system's vulnerability coverage. Future versions of the system could support other fuzzers like LibFuzzer, Radamsa, or even hybrid approaches that combine various fuzzing strategies. This would provide a more reliable platform for different use cases and binary types.

### **3. Enhanced Security Features:**

- Although two-factor authentication (2FA) was implemented to strengthen security, there is room for further enhancement. Incorporating role-based access control (RBAC) could better manage user privileges and ensure that different levels of access are granted based on the user's role (e.g., admin, developer, tester).
- Additionally, employing encryption for sensitive data both in transit and at rest would further safeguard against potential data breaches or unauthorized access.

### **4. User Interface and Experience Improvements:**

- While the system's user interface is functional, future iterations could benefit from refining the user experience. Simplifying the file upload process, providing more detailed progress feedback during fuzzing

sessions, and offering better visualizations of results would enhance usability, especially for less experienced users.

- Adding more customization options to the user interface, such as adjustable fuzzing parameters and advanced reporting features, could provide users with greater flexibility in tailoring the fuzzing process to their specific needs.

### **5. Automated Reporting Enhancements:**

- The CVE-like reports generated by the system are valuable, but additional details such as exploitability, risk scoring, or vulnerability ranking could help users prioritize mitigation efforts. Implementing integration with existing vulnerability databases, like NVD (National Vulnerability Database), could also automate the correlation of detected vulnerabilities with known issues.
- Offering more comprehensive visualizations, such as vulnerability heatmaps or trend graphs, could improve the interpretability of results for users.

### **6. Documentation and Support:**

- Comprehensive documentation and user guides should be provided to ensure that users can fully leverage the system's capabilities. This should include detailed explanations of the fuzzing process, how to interpret results, and guidance on advanced configurations.
- Implementing a support system (e.g., a FAQ section, community forum, or ticketing system) would also be beneficial for users seeking assistance or troubleshooting issues.



## 7. Optimization of Fuzzing Engines

- While Honggfuzz outperformed AFL++ in detecting vulnerabilities in picoCTF binaries, AFL++ showed better performance for custom-made test cases.
- Future development should focus on optimizing the system to support a wider variety of fuzzing engines or refining existing engines to improve performance across different use cases. This will ensure that the system can handle various types of vulnerabilities and binaries more effectively, providing consistent and reliable results in diverse scenarios.

By improving these areas, the system can grow into a stronger and more adaptable tool for automated vulnerability assessment. It would be better equipped to handle larger environments, support more use cases, and offer even stronger security and a smoother user experience.

## REFERENCES

- [1] J. Bruce. “Vulnerability management: traditional approaches vs. risk-based strategies.” LinkedIn. <https://www.linkedin.com/pulse/vulnerability-management-traditional-approaches-vs-risk-based-bruce/> (accessed August. 26, 2024).
- [2] Rebert, A., et al. “Optimizing seed selection for fuzzing.” USENIX Security Symposium. <https://www.usenix.org/system/files/conference/usenixsecurity14/sec14-paper-rebert.pdf> (accessed April. 13, 2025).
- [3] George K., et al. “Evaluating Fuzz Testing” ACM Digital Library. <https://dl.acm.org/doi/pdf/10.1145/3243734.3243804> (accessed April. 13, 2025).
- [4] Brown, T., et al. (2020). “Language Models are Few-Shot Learners.” NeurIPS. <https://arxiv.org/abs/2005.14165> (accessed April. 13, 2025).
- [5] MITRE Corporation. “Common Vulnerabilities and Exposures (CVE).” <https://cve.mitre.org> (accessed April. 13, 2025).
- [6] E. Borges. “Top 15 OSINT Tools for Expert Intelligence Gathering.” Recorded Future. <https://www.recordedfuture.com/threat-intelligence-101/tools-and-technologies/osint-tools> (accessed August. 26, 2024).
- [7] R. Dezso. “How to Use OSINT Framework Tools for Effective Pentesting.” StationX. <https://www.stationx.net/osint-framework/> (accessed August. 26, 2024).
- [8] C. Kime. “6 Top Open-Source Vulnerability Scanners & Tools.” eSecurityPlanet. <https://www.esecurityplanet.com/networks/open-source-vulnerability-scanners/> (accessed August. 26, 2024).
- [9] K. Yasar. “Network vulnerability scanning.” TechTarget. <https://www.techtarget.com/searchsecurity/definition/vulnerability-scanning> (accessed August. 26, 2024).

## REFERENCES

- [10] S. Alazmi “A Systematic Literature Review on the Characteristics and Effectiveness of Web Application Vulnerability Scanners.” IEEE.  
<https://ieeexplore.ieee.org/document/9739725> (accessed August. 26, 2024).
- [11] “The Importance of Web Application Scanning.” Acunetix.  
<https://www.acunetix.com/websitesecurity/the-importance-of-web-application-scanning/> (accessed August. 27, 2024).
- [12] “What Is the CI/CD Pipeline?.” Paloaltone Networks.  
<https://www.paloaltonetworks.com/cyberpedia/what-is-the-ci-cd-pipeline-and-ci-cd-security> (accessed August. 27, 2024).
- [13] “Static Application Security Testing.” Synopsys.  
<https://www.synopsys.com/glossary/what-is-sast.html#:~:text=Definition,known%20as%20white%20box%20testing.>  
(accessed August. 27, 2024).
- [14] G. Alvarenga. “Shift Left Security Explained.” CrowdStrike.  
<https://www.crowdstrike.com/cybersecurity-101/shift-left-security/> (accessed August. 27, 2024).
- [15] “What is Dynamic Application Security Testing (DAST)?.” OpenText.  
<https://www.opentext.com/what-is/dast> (accessed August. 27, 2024).
- [16] E. Kuzmenko. “Static Application Security Testing Explained Simply.” Kitrum.  
<https://kitrum.com/blog/static-application-security-testing-explained-simply/>  
(accessed August. 27, 2024).
- [17] “Configuration Management.” Veritis. <https://www.veritis.com/blog/chef-vs-puppet-vs-ansible-comparison-of-devops-management-tools/> (accessed August. 27, 2024).
- [18] J. Varghese. “NIST vs CIS Explained: Comparison, Benefits and Applications.” Getastra. <https://www.getastra.com/blog/compliance/nist/nist-vs-cis/#:~:text=The%20NIST%20framework%20is%20a,to%20protect%20against%20common%20cyberattacks.> (accessed August. 27, 2024).

## REFERENCES

- [19] M. McDade. “The Top 10 Container Security Tools.” ExpertInsights.  
<https://expertinsights.com/insights/the-top-container-security-tools/> (accessed August. 27, 2024).
- [20] The Cloud Native Experts. “Cloud Security Tools.” Aquasec.  
<https://www.aquasec.com/cloud-native-academy/cspm/cloud-security-tools/#:~:text=Cloud%20security%20tools%20are%20specialized,access%2C%20and%20cloud%20service%20vulnerabilities.> (accessed August. 27, 2024).
- [21] “What is Fuzz Testing and how does it work?.” Synopsys.  
<https://www.synopsys.com/glossary/what-is-fuzz-testing.html> (accessed August. 27, 2024).
- [22] “Fuzzing for software security testing and quality assurance, second edition.” Google Books,  
[https://books.google.com.my/books?hl=en&lr=&id=tKN5DwAAQBAJ&oi=fnd&pg=PR15&dq=%2BFuzzing%2Bfor%2BSoftware%2BSecurity%2BTesting%2Band%2BQuality%2BAssurance&ots=dQcCq1S979&sig=N-SrT1PIEqsfvO7qjBc3O6Rcnho&redir\\_esc=y#v=onepage&q=Fuzzing%20for%20Software%20Security%20Testing%20and%20Quality%20Assurance&f=false](https://books.google.com.my/books?hl=en&lr=&id=tKN5DwAAQBAJ&oi=fnd&pg=PR15&dq=%2BFuzzing%2Bfor%2BSoftware%2BSecurity%2BTesting%2Band%2BQuality%2BAssurance&ots=dQcCq1S979&sig=N-SrT1PIEqsfvO7qjBc3O6Rcnho&redir_esc=y#v=onepage&q=Fuzzing%20for%20Software%20Security%20Testing%20and%20Quality%20Assurance&f=false) (accessed August. 27, 2024).
- [23] V. Ganesh. “Taint-based directed whitebox fuzzing.” Researchgate.  
[https://www.researchgate.net/publication/221554473\\_Taint-based\\_Directed\\_Whitebox\\_Fuzzing](https://www.researchgate.net/publication/221554473_Taint-based_Directed_Whitebox_Fuzzing) (accessed August. 27, 2024).
- [24] “Greybox fuzzing - The fuzzing book.” Greybox Fuzzing - The Fuzzing Book.  
<https://www.fuzzingbook.org/html/GreyboxFuzzer.html> (accessed August. 27, 2024).
- [25] Google. “Google/clusterfuzz: Scalable Fuzzing Infrastructure.” GitHub.  
<https://github.com/google/clusterfuzz> (accessed August. 27, 2024).
- [26] A. Dew. “An introduction to fuzzing - what is Fuzz Testing?.” TrustInSoft, exhaustive static analysis tools for software security and safety. <https://trust-in->

## REFERENCES

- soft.com/blog/2023/05/23/an-introduction-to-fuzzing/ (accessed August. 27, 2024).
- [27] F. Rustamov. “Exploratory review of hybrid fuzzing for automated vulnerability.” Ieeexplore.  
<https://ieeexplore.ieee.org/document/9541397/references> (accessed August. 27, 2024).
- [28] R. Xu. “Symbolic execution algorithms for test generation.” <https://people.mpi-sws.org/~rupak/Papers/RuGangXuThesis.pdf> (accessed August. 28, 2024).
- [29] X. Mi, B. Wang, Y. Tang, P. Wang, and B. Yu. “Shfuzz: Selective hybrid fuzzing with branch scheduling based on binary instrumentation.” MDPI.  
<https://www.mdpi.com/2076-3417/10/16/5449> (accessed August. 28, 2024).
- [30] “Mutation-based fuzzing - The fuzzing book.” Mutation-Based Fuzzing - The Fuzzing Book. <https://www.fuzzingbook.org/html/MutationFuzzer.html> (accessed August. 28, 2024).
- [31] Y. Koike. “Bandit Optimization Framework for Mutation-Based Fuzzing.” SLOPT: Bandit Optimization Framework for mutation-based fuzzing.  
<https://dl.acm.org/doi/fullHtml/10.1145/3564625.3564659#:~:text=For%20example%2C%20mutation%2Dbased%20fuzzing,paths%2C%20and%20thus%2C%20bugs.> (accessed August. 28, 2024).
- [32] J. Li. “Comparison of generation based fuzzers and mutation.”  
[https://www.researchgate.net/figure/Comparison-of-generation-based-fuzzers-and-mutation-based-fuzzers\\_tbl2\\_325577316](https://www.researchgate.net/figure/Comparison-of-generation-based-fuzzers-and-mutation-based-fuzzers_tbl2_325577316) (accessed August. 28, 2024).
- [33] D. C. “Introduction to file format Fuzzing & Exploitation.” Medium.  
<https://danielc7.medium.com/introduction-to-file-format-fuzzing-exploitation-922143ab2ab3> (accessed August. 28, 2024).
- [34] Z. Zhang, H. Zhang, J. Zhao, and Y. Yin. “A survey on the development of network protocol fuzzing techniques.” MDPI. <https://www.mdpi.com/2079-9292/12/13/2904> (accessed August. 28, 2024).

## REFERENCES

- [35] V. Pham. “Model-based Whitebox fuzzing for program binaries.” Mboehme. <https://mboehme.github.io/paper/ASE16.pdf> (accessed August. 28, 2024).
- [36] V. Pham. “Smart Greybox Fuzzing.” Mboehme. <https://mboehme.github.io/paper/TSE19.pdf> (accessed August. 28, 2024).
- [37] P. Godefroid. “Sage: Whitebox fuzzing for Security Testing.” SAGE: Whitebox Fuzzing for Security Testing - ACM Queue. <https://queue.acm.org/detail.cfm?id=2094081> (accessed August. 28, 2024).
- [38] Wen Xu Georgia Institute of Technology et al.. “Freedom: Engineering a state-of-the-art dom fuzzer: Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security.” ACM Conferences. <https://dl.acm.org/doi/10.1145/3372297.3423340> (accessed August. 28, 2024).
- [39] GoSSIP. “FreeDom: Engineering a State-of-the-Art DOM Fuzzer.” FreeDom: Engineering a State-of-the-Art DOM Fuzzer - Group of Software Security In Progress. <https://securitygossip.com/blog/2020/12/18/freedom-engineering-a-state-of-the-art-dom-fuzzer/> (accessed August. 28, 2024).
- [40] “API fuzz testing: What is.” Aptori. <https://aptori.dev/glossary/api-fuzz-testing#:~:text=API%20Fuzz%20testing%2C%20often%20called%20API%20Fuzzing%2C%20is%20a%20dynamic,comprehensive%20testing%20of%20the%20API.> (accessed August. 28, 2024).
- [41] T. P. Ltd. “A guide to fuzz testing.” Testfully. <https://testfully.io/blog/fuzz-testing/> (accessed August. 28, 2024).
- [42] J. Wang. “Figure 1: Coverage-guided fuzzing overview.” Researchgate. [https://www.researchgate.net/figure/Coverage-guided-Fuzzing-Overview\\_fig1\\_357046806](https://www.researchgate.net/figure/Coverage-guided-Fuzzing-Overview_fig1_357046806) (accessed August. 29, 2024).
- [43] S. of Bitcoin. “Fuzzing evolution: How developers make bitcoin more secure.” Summer of Bitcoin Blog. <https://blog.summerofbitcoin.org/fuzzing-evolution-how-developers-make-bitcoin-more-secure/> (accessed August. 29, 2024).

## REFERENCES

- [44] H. Sochor, F. Ferrarotti, and D. Kaufmann. “Fuzzing-based grammar inference.” SpringerLink. [https://link.springer.com/chapter/10.1007/978-3-031-21595-7\\_6](https://link.springer.com/chapter/10.1007/978-3-031-21595-7_6) (accessed August. 29, 2024).
- [45] T. M. Jimenez. “The Mexican American Vietnam War Serviceman: The missing American.” Digitalcommons. <https://digitalcommons.calpoly.edu/cgi/viewcontent.cgi?article=2657&context=theses> (accessed August. 29, 2024).
- [46] “Common Protocol vulnerabilities.” CQR. <https://cqr.company/web-vulnerabilities/common-protocol-vulnerabilities/#:~:text=Examples%20of%20protocol%20vulnerabilities%20include,transmission%2C%20and%20insufficient%20authentication%20mechanisms.&text=The%20code%20is%20a%20simple,server%2C%20and%20receives%20a%20response.> (accessed August. 29, 2024).
- [47] Y. Wang. “Fuzzing Program Logic Deeply Hidden in Binary Program Stages.” IEEE. <https://ieeexplore.ieee.org/document/8668022> (accessed August. 29, 2024).
- [48] M. Salehi. “Discovery and Identification of Memory Corruption Vulnerabilities on Bare-Metal Embedded Devices.” IEEE. <https://ieeexplore.ieee.org/document/9707846> (accessed August. 29, 2024).
- [49] “Fuzz Testing.” Influxdata. <https://www.influxdata.com/glossary/fuzz-testing/#:~:text=Fuzz%20testing%20can%20expose%20edge,%2C%20malformed%2C%20or%20unexpected%20data.> (accessed August. 29, 2024).
- [50] S. Grob. “FUZZILLI: Fuzzing for JavaScript JIT Compiler Vulnerabilities.” NDSS. [https://www.ndss-symposium.org/wp-content/uploads/2023/02/ndss2023\\_f290\\_paper.pdf](https://www.ndss-symposium.org/wp-content/uploads/2023/02/ndss2023_f290_paper.pdf) (accessed August. 29, 2024).
- [51] “honggfuzz-rs.” DOCS.RS. <https://docs.rs/crate/honggfuzz/0.2.1> (accessed August. 29, 2024).

## REFERENCES


- [52] “Fuzzing binary-only targets.” AFLplusplus.  
[https://aflplusplus/docs/fuzzing\\_binary-only\\_targets/](https://aflplusplus/docs/fuzzing_binary-only_targets/) (accessed August. 29, 2024).
- [53] “GNATfuzz User’s Guide.” AdaCore. [https://docs.adacore.com/gnatcoverage-docs/html/gnatfuzz/gnatfuzz\\_part.html](https://docs.adacore.com/gnatcoverage-docs/html/gnatfuzz/gnatfuzz_part.html) (accessed August. 30, 2024).
- [54] Y. Li. “Principled Greybox Fuzzing: 20th International Conference on Formal Engineering Methods, ICFEM 2018, Gold Coast, QLD, Australia, November 12-16, 2018, Proceedings.” ResearchGate.  
[https://www.researchgate.net/publication/328206406\\_Principled\\_Greybox\\_Fuzzing\\_20th\\_International\\_Conference\\_on\\_Forma\\_Engineering\\_Methods\\_ICFEM\\_2018\\_Gold\\_Coast\\_QLD\\_Australia\\_November\\_12-16\\_2018\\_Proceedings#pf2](https://www.researchgate.net/publication/328206406_Principled_Greybox_Fuzzing_20th_International_Conference_on_Forma_Engineering_Methods_ICFEM_2018_Gold_Coast_QLD_Australia_November_12-16_2018_Proceedings#pf2) (accessed August. 30, 2024).
- [55] O. Jacobi. “Exploring Architectures and Capabilities of Foundational LLMs” Aporia. [https://www.aporia.com/learn/exploring-architectures-and-capabilities-of-foundational-llms/#LLM\\_architecture](https://www.aporia.com/learn/exploring-architectures-and-capabilities-of-foundational-llms/#LLM_architecture) (accessed August. 30, 2024).
- [56] “AFLplusplus.” Github. <https://github.com/AFLplusplus/AFLplusplus> (accessed August. 30, 2024).
- [57] “honggfuzz.” Github. <https://github.com/google/honggfuzz> (accessed August. 30, 2024).
- [58] “Binary Exploitation.” picoCTF. <https://play.picoctf.org/practice> (accessed August. 30, 2024).



## APPENDIX


## POSTER

# MULTI-FUZZER TECHNIQUES FOR AUTOMATED VULNERABILITIES ASSESSMENTS




Faculty of Information  
Communication and  
Technology

## 01. Introduction




With growing software complexity, traditional vulnerability detection methods fall short. This project leverages automated multi-fuzzer techniques in parallel, utilizing AFL++ and Honggfuzz to enhance software security by uncovering hidden flaws more effectively.

## 03. Objectives



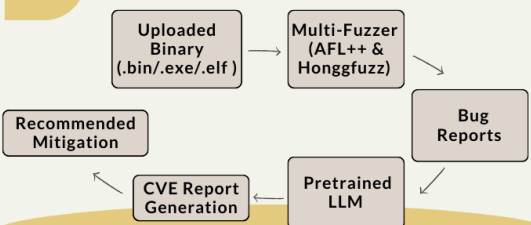
- Design a multi-fuzzing technique to automate bug and exploitation detection
- Develop a pipeline to transform bug reports into actionable vulnerabilities similar to CVE using a pretrained LLM
- Develop a framework to mitigate automatically detected vulnerabilities

## 02. Problem Statement



- Current software vulnerability testing methods lack efficiency.
- Single fuzzing tools often miss critical vulnerabilities due to their inherent limitations.

## 04. Methodology



```
graph TD; A[Uploaded Binary (.bin/.exe/.elf)] --> B[Multi-Fuzzer (AFL++ & Honggfuzz)]; B --> C[Bug Reports]; C --> D[Pretrained LLM]; D --> E[CVE Report Generation]; E --> F[Recommended Mitigation];
```

## 05. Conclusion

The integration of AFL++ and Honggfuzz improves vulnerability detection through broader coverage, while LLM-powered mitigation streamlines reporting and resolution. This approach advances automated security testing.

Project Developer: Choy Ein Jun  
Project Supervisor: Dr Aun Yichiet

## CODING WORK

### PHP Script

#### **check\_crash.php**

```
<?php
header('Content-Type: application/json');

$directory = '/home/einjun/AFLplusplus/logs';
$files = glob($directory . '/*_report.txt');
$crash_detected = false;
$latest_file = null;
$debug_info = [];

$debug_info['all_files'] = $files;

// Find the most recent report file
if (!empty($files)) {
    $latest_file = max($files);
    $debug_info['latest_file'] = $latest_file;

    $file_time = filemtime($latest_file);
    $current_time = time();
    $age = $current_time - $file_time;

    $debug_info['file_time'] = date('Y-m-d H:i:s', $file_time);
    $debug_info['current_time'] = date('Y-m-d H:i:s', $current_time);
    $debug_info['age_seconds'] = $age;

    session_start();
    $content = file_get_contents($latest_file);
    $debug_info['file_content'] = $content;

    $crash_patterns = [
        'crash',
        'crashed',
        'crashes found',
        'AFL++ crash',
        'Honggfuzz crash',
        'SEGV',
        'SIGSEGV',
        'stack overflow',
        'memory corruption'
    ];

    foreach ($crash_patterns as $pattern) {
        if (stripos($content, $pattern) !== false) {
            $crash_detected = true;
            break;
        }
    }

    $_SESSION['last_read_content'] = $content;

    $debug_info['crash_detected'] = $crash_detected;
    $debug_info['crash_patterns_checked'] = $crash_patterns;
} else {
    $debug_info['no_files_found'] = true;
}
```

## APPENDIX

```
}  
  
error_log("Crash check debug info: " . print_r($debug_info, true));  
  
echo json_encode([  
    'crash_found' => $crash_detected,  
    'latest_file' => $latest_file,  
    'debug' => $debug_info  
]);
```

**create\_user.php**

```

<?php
session_start();
$error = "";
$success = "";

$db_host = 'localhost';
$db_user = 'root';
$db_pass = "";
$db_name = 'chatbot_db';

$conn = new mysqli($db_host, $db_user, $db_pass, $db_name);

if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}

if ($_SERVER["REQUEST_METHOD"] == "POST") {
    $username = $conn->real_escape_string($_POST['username']);
    $password = $_POST['password'];
    $confirm_password = $_POST['confirm_password'];
    $email = $conn->real_escape_string($_POST['email']);

    $validation_errors = [];

    // Username validation
    if (strlen($username) < 3) {
        $validation_errors[] = "Username must be at least 3 characters long";
    } elseif (!preg_match('/^[a-zA-Z0-9_]+$/', $username)) {
        $validation_errors[] = "Username can only contain letters, numbers, and underscores";
    }

    // Password validation
    if (strlen($password) < 8) {
        $validation_errors[] = "Password must be at least 8 characters long";
    } elseif (!preg_match('/[A-Z]/', $password)) {
        $validation_errors[] = "Password must contain at least one uppercase letter";
    } elseif (!preg_match('/[a-z]/', $password)) {
        $validation_errors[] = "Password must contain at least one lowercase letter";
    } elseif (!preg_match('/[0-9]/', $password)) {
        $validation_errors[] = "Password must contain at least one number";
    } elseif (!preg_match('/^[A-Za-z0-9]/', $password)) {
        $validation_errors[] = "Password must contain at least one special character";
    }

    if ($password !== $confirm_password) {
        $validation_errors[] = "Passwords do not match";
    }

    // Email validation
    if (!filter_var($email, FILTER_VALIDATE_EMAIL)) {
        $validation_errors[] = "Invalid email format";
    }

    if (empty($validation_errors)) {
        $check_sql = "SELECT id FROM users WHERE username = ? OR email = ?";
        $check_stmt = $conn->prepare($check_sql);
        $check_stmt->bind_param("ss", $username, $email);
    }
}

```

## APPENDIX

```
$check_stmt->execute();
$result = $check_stmt->get_result();

if ($result->num_rows > 0) {
    $error = "Username or email already exists";
} else {
    // Create new user
    $hashed_password = password_hash($password, PASSWORD_DEFAULT);
    $sql = "INSERT INTO users (username, password, email) VALUES (?, ?, ?)";
    $stmt = $conn->prepare($sql);
    $stmt->bind_param("sss", $username, $hashed_password, $email);

    if ($stmt->execute()) {
        $success = "Account created successfully! Redirecting to login...";
        header("refresh:3;url=login.php");
    } else {
        $error = "Error creating account: " . $conn->error;
    }
    $stmt->close();
}
$check_stmt->close();
} else {
    $error = implode("<br>", $validation_errors);
}
}
?>

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Create Account - Multi-Fuzzer AI Chatbot System</title>
    <style>
        :root {
            --primary-color: #2c3e50;
            --secondary-color: #3498db;
            --success-color: #2ecc71;
            --error-color: #e74c3c;
            --background-color: #f4f6f9;
            --text-color: #2c3e50;
            --text-muted: #7f8c8d;
            --border-color: #e0e0e0;
            --shadow-sm: 0 2px 4px rgba(0,0,0,0.05);
            --shadow-md: 0 4px 6px rgba(0,0,0,0.1);
            --shadow-lg: 0 8px 24px rgba(0,0,0,0.1);
            --gradient-primary: linear-gradient(135deg, var(--primary-color), var(--secondary-color));
        }

        * {
            margin: 0;
            padding: 0;
            box-sizing: border-box;
        }

        body {
            font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
            background-color: var(--background-color);
            min-height: 100vh;
        }
    </style>
</head>
```

## APPENDIX

```
display: flex;
align-items: center;
justify-content: center;
padding: 20px;
color: var(--text-color);
line-height: 1.6;
position: relative;
overflow-y: auto;
overflow-x: hidden;
}

.background-pattern {
position: fixed;
top: 0;
left: 0;
width: 100%;
height: 100%;
background-image:
  radial-gradient(circle at 25% 25%, rgba(52, 152, 219, 0.1) 0%, transparent 50%),
  radial-gradient(circle at 75% 75%, rgba(44, 62, 80, 0.1) 0%, transparent 50%);
z-index: -1;
}

.register-container {
background: white;
padding: 2.5rem;
border-radius: 16px;
box-shadow: var(--shadow-lg);
width: 100%;
max-width: 420px;
position: relative;
overflow: hidden;
backdrop-filter: blur(10px);
border: 1px solid rgba(255, 255, 255, 0.2);
animation: slideUp 0.5s ease-out;
}

@keyframes slideUp {
from {
opacity: 0;
transform: translateY(20px);
}
to {
opacity: 1;
transform: translateY(0);
}
}

.register-header {
text-align: center;
margin-bottom: 2rem;
position: relative;
}

.register-header h1 {
color: var(--text-color);
font-size: 1.8rem;
margin-bottom: 0.5rem;
display: flex;
```

## APPENDIX

```
    align-items: center;
    justify-content: center;
    gap: 0.8rem;
}

.register-header svg {
    color: var(--secondary-color);
    filter: drop-shadow(0 2px 4px rgba(52, 152, 219, 0.2));
}

.register-header p {
    color: var(--text-muted);
    font-size: 0.95rem;
    margin-top: 0.5rem;
}

.form-group {
    margin-bottom: 1.8rem;
    position: relative;
}

.form-group label {
    display: block;
    margin-bottom: 0.8rem;
    color: var(--text-color);
    font-weight: 500;
    font-size: 0.95rem;
    transition: color 0.3s ease;
}

.form-group input {
    width: 100%;
    padding: 1rem;
    border: 2px solid var(--border-color);
    border-radius: 12px;
    font-size: 1rem;
    transition: all 0.3s ease;
    background: #f8f9fa;
    color: var(--text-color);
}

.form-group input:focus {
    border-color: var(--secondary-color);
    outline: none;
    box-shadow: 0 0 0 4px rgba(52, 152, 219, 0.1);
    background: white;
}

.form-group input::placeholder {
    color: var(--text-muted);
    opacity: 0.5;
}

.password-strength {
    margin-top: 0.5rem;
    font-size: 0.85rem;
    color: var(--text-muted);
}
```

## APPENDIX

```
.strength-meter {
  height: 4px;
  background: #eee;
  margin-top: 0.5rem;
  border-radius: 2px;
  overflow: hidden;
}

.strength-meter div {
  height: 100%;
  width: 0;
  transition: all 0.3s ease;
}

.requirements-list {
  list-style: none;
  padding: 0;
  margin: 10px 0;
  font-size: 0.85rem;
  color: var(--text-muted);
}

.requirement-item {
  display: flex;
  align-items: center;
  margin: 5px 0;
  transition: color 0.3s ease;
}

.requirement-item.valid {
  color: var(--success-color);
}

.requirement-item::before {
  content: "•";
  margin-right: 8px;
  color: var(--error-color);
}

.requirement-item.valid::before {
  content: "✓";
  color: var(--success-color);
}

.error-message {
  background-color: rgba(231, 76, 60, 0.1);
  color: var(--error-color);
  padding: 1rem;
  border-radius: 12px;
  margin-bottom: 1.5rem;
  font-size: 0.95rem;
  display: flex;
  align-items: center;
  gap: 0.8rem;
  border: 1px solid rgba(231, 76, 60, 0.2);
  animation: shake 0.5s ease-in-out;
}

@keyframes shake {
```



## APPENDIX

```
0%, 100% { transform: translateX(0); }
25% { transform: translateX(-5px); }
75% { transform: translateX(5px); }
}

.success-message {
  background-color: rgba(46, 204, 113, 0.1);
  color: var(--success-color);
  padding: 1rem;
  border-radius: 12px;
  margin-bottom: 1.5rem;
  font-size: 0.95rem;
  display: flex;
  align-items: center;
  gap: 0.8rem;
  border: 1px solid rgba(46, 204, 113, 0.2);
}

.register-button {
  width: 100%;
  padding: 1rem;
  background: var(--gradient-primary);
  color: white;
  border: none;
  border-radius: 12px;
  font-size: 1rem;
  font-weight: 600;
  cursor: pointer;
  transition: all 0.3s ease;
  position: relative;
  overflow: hidden;
  margin-bottom: 1.5rem;
}

.register-button:hover {
  transform: translateY(-2px);
  box-shadow: 0 4px 12px rgba(52, 152, 219, 0.2);
}

.register-button:active {
  transform: translateY(1px);
}

.register-button::after {
  content: "";
  position: absolute;
  top: 0;
  left: 0;
  width: 100%;
  height: 100%;
  background: linear-gradient(45deg, transparent, rgba(255,255,255,0.2), transparent);
  transform: translateX(-100%);
  transition: transform 0.6s ease;
}

.register-button:hover::after {
  transform: translateX(100%);
}
```

## APPENDIX

```
.login-link {
  text-align: center;
  color: var(--text-muted);
  font-size: 0.95rem;
}

.login-link a {
  color: var(--secondary-color);
  text-decoration: none;
  font-weight: 500;
  transition: color 0.3s ease;
  position: relative;
}

.login-link a::after {
  content: "";
  position: absolute;
  width: 100%;
  height: 2px;
  bottom: -2px;
  left: 0;
  background-color: var(--secondary-color);
  transform: scaleX(0);
  transform-origin: right;
  transition: transform 0.3s ease;
}

.login-link a:hover {
  color: var(--primary-color);
}

.login-link a:hover::after {
  transform: scaleX(1);
  transform-origin: left;
}

.decoration {
  position: absolute;
  width: 300px;
  height: 300px;
  background: linear-gradient(45deg, rgba(52, 152, 219, 0.1), rgba(44, 62, 80, 0.1));
  border-radius: 50%;
  z-index: 0;
  animation: float 6s ease-in-out infinite;
}

@keyframes float {
  0%, 100% { transform: translateY(0); }
  50% { transform: translateY(-20px); }
}

.decoration-1 {
  top: -150px;
  right: -150px;
  animation-delay: 0s;
}

.decoration-2 {
  bottom: -150px;
```

## APPENDIX

```
    left: -150px;
    animation-delay: -3s;
  }

  @media (max-width: 480px) {
    .register-container {
      margin: 1rem;
      padding: 1.5rem;
    }

    .register-header h1 {
      font-size: 1.5rem;
    }

    .form-group input {
      font-size: 0.95rem;
      padding: 0.8rem;
    }
  }
</style>
</head>
<body>
  <div class="background-pattern"></div>
  <div class="register-container">
    <div class="decoration decoration-1"></div>
    <div class="decoration decoration-2"></div>

    <div class="register-header">
      <h1>
        <svg style="width:28px;height:28px" viewBox="0 0 24 24">
          <path fill="currentColor" d="M12,4A4,4 0 0,1 16,8A4,4 0 0,1 12,12A4,4 0 0,1 8,8A4,4 0
0,1 12,4M12,14C16.42,14 20,15.79 20,18V20H4V18C4,15.79 7.58,14 12,14Z" />
        </svg>
        Create Account
      </h1>
      <p>Join Multi-Fuzzer AI Chatbot System</p>
    </div>

    <?php if ($error): ?>
      <div class="error-message">
        <svg style="width:20px;height:20px" viewBox="0 0 24 24">
          <path fill="currentColor" d="M13 14H11V9H13M13 18H11V16H13M1 21H23L12 2L1
21Z" />
        </svg>
        <?php echo $error; ?>
      </div>
    <?php endif; ?>

    <?php if ($success): ?>
      <div class="success-message">
        <svg style="width:20px;height:20px" viewBox="0 0 24 24">
          <path fill="currentColor" d="M12 2C6.5 2 2 6.5 2 12S6.5 22 12 22 17.5 22 17.5 2
12 2M10 17L5 12L6.41 10.59L10 14.17L17.59 6.58L19 8L10 17Z" />
        </svg>
        <?php echo $success; ?>
      </div>
    <?php endif; ?>
```

```

<form method="POST" action="<?php echo htmlspecialchars($_SERVER["PHP_SELF"]); ?>"
id="registerForm">
  <div class="form-group">
    <label for="username">Username</label>
    <input type="text" id="username" name="username" required minlength="3"
      pattern="[a-zA-Z0-9_]+" autocomplete="username"
      placeholder="Choose a username">
    <div class="password-strength">
      <div class="strength-meter">
        <div id="username-strength-bar"></div>
      </div>
      <span id="username-strength-text">Username strength: Too weak</span>
    </div>
  </div>

  <div class="form-group">
    <label for="email">Email</label>
    <input type="email" id="email" name="email" required
      autocomplete="email" placeholder="Enter your email">
  </div>

  <div class="form-group">
    <label for="password">Password</label>
    <input type="password" id="password" name="password" required
      minlength="8" autocomplete="new-password"
      placeholder="Create a strong password">
    <div class="password-strength">
      <div class="strength-meter">
        <div id="password-strength-bar"></div>
      </div>
      <span id="password-strength-text">Password strength: Too weak</span>
      <ul id="password-requirements" class="requirements-list">
        <li class="requirement-item" id="length">At least 8 characters</li>
        <li class="requirement-item" id="uppercase">One uppercase letter</li>
        <li class="requirement-item" id="lowercase">One lowercase letter</li>
        <li class="requirement-item" id="number">One number</li>
        <li class="requirement-item" id="special">One special character</li>
      </ul>
    </div>
  </div>

  <div class="form-group">
    <label for="confirm_password">Confirm Password</label>
    <input type="password" id="confirm_password" name="confirm_password"
      required minlength="8" autocomplete="new-password"
      placeholder="Confirm your password">
  </div>

  <button type="submit" class="register-button">Create Account</button>
</form>

<div class="login-link">
  Already have an account? <a href="login.php">Login here</a>
</div>
</div>

<script>
  function checkPasswordStrength(password) {
    let strength = 0;

```

```

const requirements = {
  length: password.length >= 8,
  uppercase: /[A-Z]/.test(password),
  lowercase: /[a-z]/.test(password),
  number: /[0-9]/.test(password),
  special: /^[^A-Za-z0-9]/.test(password)
};

Object.keys(requirements).forEach(req => {
  const element = document.getElementById(req);
  if (requirements[req]) {
    element.classList.add('valid');
  } else {
    element.classList.remove('valid');
  }
});

strength = Object.values(requirements).filter(Boolean).length * 20;
const strengthBar = document.getElementById('password-strength-bar');
const strengthText = document.getElementById('password-strength-text');

strengthBar.style.width = strength + '%';
strengthBar.style.backgroundColor =
  strength <= 20 ? '#e74c3c' :
  strength <= 40 ? '#f39c12' :
  strength <= 60 ? '#f1c40f' :
  strength <= 80 ? '#3498db' :
  '#2ecc71';

strengthText.innerHTML = `Password strength: <strong>${
  strength <= 20 ? 'Very Weak' :
  strength <= 40 ? 'Weak' :
  strength <= 60 ? 'Medium' :
  strength <= 80 ? 'Strong' :
  'Very Strong'
}</strong>`;

return strength;
}

function checkUsernameStrength(username) {
  let strength = 0;
  if (username.length >= 3) strength += 25;
  if (username.length >= 5) strength += 25;
  if (username.length >= 8) strength += 25;
  if (/[A-Z]/.test(username)) strength += 25;

  const strengthBar = document.getElementById('username-strength-bar');
  const strengthText = document.getElementById('username-strength-text');

  strengthBar.style.width = strength + '%';
  strengthBar.style.backgroundColor =
    strength <= 25 ? '#e74c3c' :
    strength <= 50 ? '#f39c12' :
    strength <= 75 ? '#f1c40f' :
    '#2ecc71';

  strengthText.innerHTML = `Username strength: <strong>${
    strength <= 25 ? 'Weak' :

```

```

        strength <= 50 ? 'Medium' :
        strength <= 75 ? 'Strong' :
        'Very Strong'
    }</strong>`;

    return strength;
}

// Real-time validation
document.getElementById('password').addEventListener('input', function(e) {
    checkPasswordStrength(e.target.value);
});

document.getElementById('username').addEventListener('input', function(e) {
    checkUsernameStrength(e.target.value);
});

// Form validation
document.getElementById('registerForm').addEventListener('submit', function(e) {
    const password = document.getElementById('password').value;
    const confirmPassword = document.getElementById('confirm_password').value;
    const username = document.getElementById('username').value;
    const email = document.getElementById('email').value;

    let isValid = true;
    let errorMessage = "";

    // Username validation
    if (username.length < 3) {
        isValid = false;
        errorMessage += 'Username must be at least 3 characters long\n';
    }
    if (!/^[a-zA-Z0-9_]+$/.test(username)) {
        isValid = false;
        errorMessage += 'Username can only contain letters, numbers, and underscores\n';
    }

    // Password validation
    if (password.length < 8) {
        isValid = false;
        errorMessage += 'Password must be at least 8 characters long\n';
    }
    if (!/[A-Z]/.test(password)) {
        isValid = false;
        errorMessage += 'Password must contain at least one uppercase letter\n';
    }
    if (!/[a-z]/.test(password)) {
        isValid = false;
        errorMessage += 'Password must contain at least one lowercase letter\n';
    }
    if (!/[0-9]/.test(password)) {
        isValid = false;
        errorMessage += 'Password must contain at least one number\n';
    }
    if (!/^[^A-Za-z0-9]/.test(password)) {
        isValid = false;
        errorMessage += 'Password must contain at least one special character\n';
    }
}

```

## APPENDIX

```
        if (password !== confirmPassword) {
            isValid = false;
            errorMessage += 'Passwords do not match\n';
        }

        // Email validation
        if (!/^^[^\\s@]+@[^\\s@]+\\.^[^\\s@]+$/i.test(email)) {
            isValid = false;
            errorMessage += 'Please enter a valid email address\n';
        }

        if (!isValid) {
            e.preventDefault();
            alert(errorMessage);
            return false;
        }

        return true;
    });

    document.querySelectorAll('.form-group input').forEach(input => {
        input.addEventListener('focus', function() {
            this.parentElement.classList.add('focused');
        });

        input.addEventListener('blur', function() {
            this.parentElement.classList.remove('focused');
        });
    });
</script>
</body>
</html>
```

**index.php**

```

<?php
session_start();
if (!isset($_SESSION['user_id'])) {
    header("Location: login.php");
    exit();
}

$logsDir = '/home/einjun/AFLplusplus/logs';

if (!isset($_SESSION['logs_cleaned'])) {
    if (is_dir($logsDir)) {
        $files = glob($logsDir . '/*');
        foreach ($files as $file) {
            if (is_file($file)) {
                unlink($file);
            }
        }
    }
    $_SESSION['logs_cleaned'] = true;
}

error_reporting(E_ALL);
ini_set('display_errors', 1);

if (!function_exists('curl_init')) {
    die('cURL is not installed on this server!');
}

$db_host = 'localhost';
$db_user = 'root';
$db_pass = '';
$db_name = 'chatbot_db';

$conn = new mysqli($db_host, $db_user, $db_pass, $db_name);

if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}

define('OPENAI_API_KEY', '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');

if (!file_exists('uploads')) {
    mkdir('uploads', 0777, true);
}

if (!isset($_SESSION['chat_history'])) {
    $_SESSION['chat_history'] = [];
}

// Handle file upload
if (isset($_FILES['security_file'])) {
    $target_dir = "/home/einjun/AFLplusplus/";
    $target_file = $target_dir . basename($_FILES["security_file"]["name"]);
    $uploadOk = 1;

    if ($_FILES["security_file"]["size"] > 500000) {
        echo "Sorry, your file is too large.";
    }
}

```



## APPENDIX

```
$uploadOk = 0;
}

if ($uploadOk == 1) {
    if (move_uploaded_file($_FILES["security_file"]["tmp_name"], $target_file)) {
        chmod($target_file, 0777);
        $filename = pathinfo($_FILES["security_file"]["name"], PATHINFO_FILENAME);
        header("Location: index1.php?command=RUN_PARALLEL 30 " . urlencode($filename));
        exit;
    } else {
        echo "Sorry, there was an error uploading your file.";
    }
}
}

// Get the latest parallel fuzzing report before sending to LLM
function getLatestFuzzingReport() {
    $directory = '/home/einjun/AFLplusplus/logs';
    $files = glob($directory . '/parallel_fuzzing_*_report.txt');

    if (!empty($files)) {
        $latest_file = max($files);
        return file_get_contents($latest_file);
    }
    return null;
}

if (isset($_POST['message'])) {
    $user_message = $conn->real_escape_string($_POST['message']);
    $timestamp = date('Y-m-d H:i:s');

    if (!isset($_SESSION['crash_analyzed'])) {
        $report_content = getLatestFuzzingReport();
        if ($report_content) {
            $user_message = "Please analyze this fuzzing crash report for potential vulnerabilities:\n\n" .
                $report_content;
            $_SESSION['crash_analyzed'] = true;
        }
    }
    $response = callOpenAI($user_message);

    header('Content-Type: application/json');
    echo json_encode(['response' => $response]);
    exit;
}

if ($_SERVER['REQUEST_METHOD'] === 'GET') {
    unset($_SESSION['crash_analyzed']);
    clearChatHistory();
}

function callOpenAI($message) {
    $url = 'https://api.openai.com/v1/chat/completions';

    $headers = [
        'Content-Type: application/json',
        'Authorization: Bearer ' . OPENAI_API_KEY
    ];
```

## APPENDIX

```
$system_message = [  
  'role' => 'system',  
  'content' => 'You are a multi-fuzzer report AI assistant. Analyze the fuzzing crash report, provide  
CVE-like report and provide mitigation recommendations in the following format for each  
vulnerability. Elaborate more professionally by adding more information. Format my response into  
more structured way (Separate each part):'
```

```
===== CVE Report
```

```
=====
```

```
Program Name      : [Program name]  
Vulnerability Name : [Short name/title of vulnerability]  
Description       : [Brief but clear description of the issue and how it occurs]  
Impact           : [Describe the effect and consequence if exploited]  
CVSS Score        : [If applicable, provide score out of 10]  
Affected Component : [Component, function, or module affected]
```

```
=====
```

```
=====
```

```
Recommendations:
```

```
Code Changes Required: [Describe exact or general code-level changes to prevent the issue]
```

```
=====
```

```
=====
```

```
Mitigation Link:
```

```
Action: install_[tool_name_below]
```

For each Action, use one of these predefined installation commands, choose only the most suitable one based on the vulnerabilities:

- Action: install\_apparmor (for AppArmor security)
- Action: install\_asan (for AddressSanitizer)
- Action: install\_hardened\_malloc (for HardenedMalloc)
- Action: install\_exploit\_mitigations (for general exploit mitigations)
- Action: install\_disk\_flooding\_attack\_mitigation (for disk flooding attacks)
- Action: install\_inodes\_exhaustion\_mitigation (for inodes exhaustion attacks)

Each Action should be on a new line and start with "Action: install\_" do not mention the brackets, only install 1 of them at once.

Instruction:Elaborate more on the effect of installing the tool. Explain what tool should be installed and why it helps.Include those "===" for formatting purpose.

```
=====
```

```
=====
```

```
,
```

```
];
```

```
// Preserve conversation history
```

```
$messages = [$system_message];
```

```
// Add conversation history
```

```
foreach ($_SESSION['chat_history'] as $chat) {
```

```
  $messages[] = [  
    'role' => $chat['is_bot'] ? 'assistant' : 'user',  
    'content' => $chat['message']  
  ];
```

```
  }
```

```
}
```

```
$messages[] = [  
  'role' => 'user',  
  'content' => $message  
];
```

```
];
```

```
];
```

## APPENDIX

```
$data = [  
  'model' => 'gpt-3.5-turbo',  
  'messages' => $messages,  
  'temperature' => 0.7,  
  'max_tokens' => 2000,  
  'presence_penalty' => 0.3,  
  'frequency_penalty' => 0.3  
];  
  
$ch = curl_init($url);  
curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);  
curl_setopt($ch, CURLOPT_POST, true);  
curl_setopt($ch, CURLOPT_POSTFIELDS, json_encode($data));  
curl_setopt($ch, CURLOPT_HTTPHEADER, $headers);  
curl_setopt($ch, CURLOPT_SSL_VERIFYPEER, false);  
curl_setopt($ch, CURLOPT_SSL_VERIFYHOST, 0);  
$response = curl_exec($ch);  
  
if (curl_errno($ch)) {  
  error_log('Curl error: ' . curl_error($ch));  
  return 'Error: ' . curl_error($ch);  
}  
  
curl_close($ch);  
$response_data = json_decode($response, true);  
error_log('OpenAI Response: ' . print_r($response_data, true));  
  
if (isset($response_data['error'])) {  
  error_log('OpenAI Error: ' . print_r($response_data['error'], true));  
  return 'Error: ' . $response_data['error']['message'];  
}  
  
if (isset($response_data['choices'][0]['message']['content'])) {  
  $_SESSION['chat_history'][] = [  
    'message' => $message,  
    'is_bot' => false  
  ];  
  $_SESSION['chat_history'][] = [  
    'message' => $response_data['choices'][0]['message']['content'],  
    'is_bot' => true  
  ];  
  
  $formatted_response = formatResponse($response_data['choices'][0]['message']['content']);  
  return $formatted_response;  
} else {  
  return 'Sorry, I could not process your request. Please check the error logs.';  
}  
}  
  
function formatResponse($response) {  
  $formatted = "  
  <style>  
    .modal {  
      display: none;  
      position: fixed;  
      top: 0;  
      left: 0;  
      width: 100%;  
    }  
  ";
```

## APPENDIX

```
    height: 100%;
    background-color: rgba(0,0,0,0.5);
    z-index: 1000;
  }
  .modal-content {
    background-color: #fefefe;
    margin: 15% auto;
    padding: 20px;
    border: 1px solid #888;
    width: 80%;
    max-width: 500px;
    border-radius: 8px;
  }
  .modal-buttons {
    display: flex;
    justify-content: flex-end;
    gap: 15px;
    margin-top: 25px;
  }
  .modal-btn {
    padding: 12px 30px;
    font-size: 16px;
    font-weight: 600;
    border: none;
    border-radius: 8px;
    cursor: pointer;
    transition: all 0.3s ease;
    min-width: 120px;
    text-transform: uppercase;
    letter-spacing: 0.5px;
  }
  .cancel-btn {
    background-color: #f8f9fa;
    color: #dc3545;
    border: 2px solid #dc3545;
  }
  .cancel-btn:hover {
    background-color: #dc3545;
    color: white;
    transform: translateY(-2px);
    box-shadow: 0 3px 8px rgba(220, 53, 69, 0.3);
  }
  .confirm-btn {
    background-color: #28a745;
    color: white;
    border: 2px solid #28a745;
  }
  .confirm-btn:hover {
    background-color: #218838;
    border-color: #218838;
    transform: translateY(-2px);
    box-shadow: 0 3px 8px rgba(40, 167, 69, 0.3);
  }
  .modal-btn:active {
    transform: translateY(0);
    box-shadow: none;
  }
}
</style>";
```

```

$formatted .= "
<div id='installModal' class='modal'>
  <div class='modal-content'>
    <h3>Confirm Installation</h3>
    <p id='modalText'>Are you sure you want to install this security tool?</p>
    <div class='modal-buttons'>
      <button onclick='cancelInstall()' class='modal-btn cancel-btn'>Cancel</button>
      <button id='confirmInstall' class='modal-btn confirm-btn'>Install</button>
    </div>
  </div>
</div>";

$sections = explode("\n", $response);
$formatted .= "<div style='font-family: Arial, sans-serif; line-height: 1.6;'>";

// Check if response contains keywords
$memoryAllocationDetected = (stripos($response, 'testdisk') !== false);
$nodesExhaustionDetected = (stripos($response, 'testfile') !== false);

foreach ($sections as $section) {
  $trimmed = trim($section);

  if (preg_match('/^-\?s*Action:\s*install_(\w+)$/i', $trimmed, $matches)) {
    $tool = $matches[1];
    $formatted .= "<div class='security-button-container'>
      <button
        class='security-button'
        onclick='showInstallConfirmation(\"$tool\")'
      >
        <span class='button-content'>
          <svg class='install-icon' viewBox='0 0 24 24'>
            <path fill='currentColor' d='M19.35 10.04C18.67 6.59 15.64 4 12 4 9.11 4 6.6 5.64
5.35 8.04 2.34 8.36 0 10.91 0 14c0 3.31 2.69 6 6 6h13c2.76 0 5-2.24 5-5 0-2.64-2.05-4.78-4.65-
4.96zM17 13l-5 5-5h3V9h4v4h3z'/>
          </svg>
          Install " . ucfirst($tool) . "
        </span>
      </button>
    </div>
    <style>
    .security-button-container {
      display: flex;
      justify-content: center;
      margin: 20px 0;
    }

    .security-button {
      background: linear-gradient(45deg, #2196F3, #00BCD4);
      border: none;
      border-radius: 25px;
      color: white;
      cursor: pointer;
      font-size: 16px;
      font-weight: 600;
      padding: 12px 30px;
      position: relative;
      overflow: hidden;
      transition: all 0.3s ease;
      box-shadow: 0 4px 15px rgba(0, 0, 0, 0.2);
    }
  </style>
  </div>";
  }
}

```

## APPENDIX

```
}

.security-button:hover {
  transform: translateY(-2px);
  box-shadow: 0 6px 20px rgba(33, 150, 243, 0.3);
  background: linear-gradient(45deg, #00BCD4, #2196F3);
}

.security-button:active {
  transform: translateY(1px);
  box-shadow: 0 2px 10px rgba(33, 150, 243, 0.3);
}

.security-button::before {
  content: "";
  position: absolute;
  top: 0;
  left: -100%;
  width: 100%;
  height: 100%;
  background: linear-gradient(
    120deg,
    transparent,
    rgba(255, 255, 255, 0.3),
    transparent
  );
  transition: 0.5s;
}

.security-button:hover::before {
  left: 100%;
}

.button-content {
  display: flex;
  align-items: center;
  gap: 8px;
}

.install-icon {
  width: 20px;
  height: 20px;
  transition: transform 0.3s ease;
}

.security-button:hover .install-icon {
  transform: scale(1.1);
}

@keyframes pulse {
  0% {
    box-shadow: 0 0 0 0 rgba(33, 150, 243, 0.4);
  }
  70% {
    box-shadow: 0 0 0 10px rgba(33, 150, 243, 0);
  }
  100% {
    box-shadow: 0 0 0 0 rgba(33, 150, 243, 0);
  }
}
```

```

    }

    .security-button:focus {
        outline: none;
        animation: pulse 1.5s infinite;
    }
</style>";
} else {
    if (!empty($trimmed)) {
        if (strpos($trimmed, '**') !== false) {
            $trimmed = preg_replace('/\*\*(.*?)\*\*/', '<strong>$1</strong>', $trimmed);
        }
        $formatted .= "<p>$trimmed</p>";
    }
}
}

if ($memoryAllocationDetected) {
    $formatted .= "<div class='security-button-container'>
        <button
            class='security-button'
            onclick='showInstallConfirmation(\"disk_flooding_attack_mitigation\")'
        >
            <span class='button-content'>
                <svg class='install-icon' viewBox='0 0 24 24'>
                    <path fill='currentColor' d='M19.35 10.04C18.67 6.59 15.64 4 12 4 9.11 4 6.6 5.64 5.35
8.04 2.34 8.36 0 10.91 0 14c0 3.31 2.69 6 6 6h13c2.76 0 5-2.24 5-5 0-2.64-2.05-4.78-4.65-4.96zM17
13l-5 5-5-5h3V9h4v4h3z'/>
                </svg>
                Install Disk Flooding Attack Mitigation
            </span>
        </button>
    </div>";
}

if ($nodesExhaustionDetected) {
    $formatted .= "<div class='security-button-container'>
        <button
            class='security-button'
            onclick='showInstallConfirmation(\"inodes_exhaustion_mitigation\")'
        >
            <span class='button-content'>
                <svg class='install-icon' viewBox='0 0 24 24'>
                    <path fill='currentColor' d='M19.35 10.04C18.67 6.59 15.64 4 12 4 9.11 4 6.6 5.64 5.35
8.04 2.34 8.36 0 10.91 0 14c0 3.31 2.69 6 6 6h13c2.76 0 5-2.24 5-5 0-2.64-2.05-4.78-4.65-4.96zM17
13l-5 5-5-5h3V9h4v4h3z'/>
                </svg>
                Install Inodes Exhaustion Mitigation
            </span>
        </button>
    </div>";
}

$formatted .= "</div>";
return $formatted;
}

function clearChatHistory() {
    $_SESSION['chat_history'] = [];
}

```

## APPENDIX

```
}

if ($_SERVER['REQUEST_METHOD'] === 'GET') {
    clearChatHistory();
}
?>

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Multi-Fuzzer AI Chatbot System</title>
    <style>
        :root {
            --primary-color: #2c3e50;
            --secondary-color: #3498db;
            --accent-color: #e74c3c;
            --background-color: #f4f6f9;
            --chat-bg: #ffffff;
            --text-color: #2c3e50;
            --border-radius: 8px;
            --shadow-color: rgba(0, 0, 0, 0.1);
        }

        body {
            font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
            margin: 0;
            padding: 0;
            background-color: var(--background-color);
            height: 100vh;
            display: flex;
            align-items: center;
            justify-content: center;
            color: var(--text-color);
        }

        .chat-container {
            width: 90%;
            max-width: 1200px;
            height: 90vh;
            margin: 0 auto;
            background-color: var(--chat-bg);
            border-radius: var(--border-radius);
            box-shadow: 0 4px 20px var(--shadow-color);
            display: flex;
            flex-direction: column;
            padding: 2rem;
            position: relative;
            transition: box-shadow 0.3s ease;
        }

        .chat-container:hover {
            box-shadow: 0 8px 40px var(--shadow-color);
        }

        h1 {
            color: var(--primary-color);
            margin: 0 0 1.5rem 0;
        }
```



## APPENDIX

```
    font-size: 2rem;
    font-weight: 700;
    text-align: center;
    text-transform: uppercase;
}

.file-upload {
    background: linear-gradient(145deg, #ffffff, #f8f9fa);
    border: 2px dashed var(--secondary-color);
    border-radius: var(--border-radius);
    padding: 1rem;
    margin-bottom: 1.5rem;
    transition: all 0.3s ease;
}

.file-upload:hover {
    border-color: var(--accent-color);
    transform: translateY(-2px);
}

.file-upload h3 {
    color: var(--primary-color);
    margin: 0 0 1rem 0;
    font-size: 1.3rem;
    font-weight: 600;
}

.file-upload form {
    display: flex;
    gap: 1rem;
    justify-content: center;
    align-items: center;
    flex-wrap: wrap;
}

input[type="file"] {
    background-color: white;
    padding: 0.5rem;
    border-radius: var(--border-radius);
    border: 1px solid #ddd;
    transition: border-color 0.3s ease;
}

input[type="file"]:focus {
    border-color: var(--secondary-color);
    outline: none;
}

input[type="submit"] {
    background-color: var(--secondary-color);
    color: white;
    border: none;
    padding: 0.8rem 1.5rem;
    border-radius: var(--border-radius);
    cursor: pointer;
    transition: all 0.3s ease;
    font-weight: 600;
}
```

## APPENDIX

```
input[type="submit"]:hover {
  background-color: var(--primary-color);
  transform: translateY(-2px);
}

.chat-box {
  flex: 1;
  background-color: #ffffff;
  border-radius: var(--border-radius);
  padding: 1.5rem;
  margin-bottom: 1.5rem;
  overflow-y: auto;
  box-shadow: 0 2px 12px rgba(0, 0, 0, 0.08);
  border: 1px solid #e6e9ef;
}

.message {
  display: flex;
  margin-bottom: 25px;
  animation: fadeIn 0.3s ease;
  max-width: 90%;
}

.message.user {
  justify-content: flex-end;
  margin-left: auto;
}

.message.bot {
  justify-content: flex-start;
  margin-right: auto;
}

.message-content {
  max-width: 85%;
  padding: 16px 20px;
  border-radius: var(--border-radius);
  font-size: 15px;
  line-height: 1.6;
  position: relative;
  box-shadow: 0 2px 8px rgba(0, 0, 0, 0.08);
}

.bot .message-content {
  background: #f8f9fa;
  color: #2c3e50;
  border: 1px solid #e9ecef;
  border-left: 4px solid var(--secondary-color);
}

.bot .message-content h2 {
  font-size: 18px;
  color: #2c3e50;
  margin: 15px 0 10px;
  padding-bottom: 8px;
  border-bottom: 1px solid #eee;
}

.bot .message-content h3 {
```

## APPENDIX

```
font-size: 16px;
color: #34495e;
margin: 12px 0 8px;
}

.bot .message-content ul,
.bot .message-content ol {
margin: 10px 0;
padding-left: 20px;
}

.bot .message-content li {
margin: 8px 0;
line-height: 1.5;
}

.bot .message-content pre {
background: #f8f9fa;
border: 1px solid #e9ecef;
border-left: 3px solid #3498db;
color: #2c3e50;
padding: 15px;
border-radius: 6px;
margin: 12px 0;
font-family: 'Monaco', 'Menlo', 'Ubuntu Mono', monospace;
font-size: 14px;
overflow-x: auto;
}

.bot .message-content code {
background: #f8f9fa;
color: #e83e8c;
padding: 2px 6px;
border-radius: 4px;
font-family: 'Monaco', 'Menlo', 'Ubuntu Mono', monospace;
font-size: 14px;
}

.bot .message-content .important {
background: #fff3cd;
border: 1px solid #ffeeba;
border-left: 4px solid #ffc107;
padding: 12px 15px;
margin: 10px 0;
border-radius: 6px;
}

.bot .message-content .vulnerability {
background: #f8f9fa;
border: 1px solid #e9ecef;
border-radius: 8px;
padding: 15px;
margin: 12px 0;
}

.bot .message-content .vulnerability-header {
font-weight: 600;
color: #2c3e50;
margin-bottom: 8px;
}
```

## APPENDIX

```
}

.bot .message-content .recommendation {
  background: #e8f4fd;
  border: 1px solid #bee5eb;
  border-radius: 8px;
  padding: 15px;
  margin: 12px 0;
}

.bot .message-content .action {
  background: #f1f8ff;
  border: 1px solid #c8e1ff;
  border-radius: 6px;
  padding: 10px 15px;
  margin: 8px 0;
  color: #0366d6;
  font-weight: 500;
}

.timestamp {
  font-size: 12px;
  color: #6c757d;
  margin-top: 8px;
  text-align: right;
}

.avatar {
  width: 40px;
  height: 40px;
  border-radius: 50%;
  margin: 0 12px;
  display: flex;
  align-items: center;
  justify-content: center;
  font-size: 18px;
  flex-shrink: 0;
  box-shadow: 0 2px 6px rgba(0, 0, 0, 0.1);
}

.bot .avatar {
  background: #f8f9fa;
  border: 2px solid #e9ecef;
  color: #2c3e50;
}

.bot .message-content a {
  color: #3498db;
  text-decoration: none;
  border-bottom: 1px dotted #3498db;
  transition: all 0.2s ease;
}

.bot .message-content a:hover {
  color: #2980b9;
  border-bottom-style: solid;
}

.bot .message-content table {
```

## APPENDIX

```
width: 100%;
border-collapse: collapse;
margin: 12px 0;
}

.bot .message-content th,
.bot .message-content td {
padding: 10px;
border: 1px solid #e9ecef;
text-align: left;
}

.bot .message-content th {
background: #f8f9fa;
font-weight: 600;
}

.bot .message-content ul.checklist {
list-style: none;
padding-left: 0;
}

.bot .message-content ul.checklist li {
position: relative;
padding-left: 25px;
margin: 8px 0;
}

.bot .message-content ul.checklist li:before {
content: "✓";
position: absolute;
left: 0;
color: #28a745;
}

.typing-indicator {
display: flex;
align-items: center;
margin-left: 58px;
margin-bottom: 20px;
}

.typing-dot {
width: 8px;
height: 8px;
margin: 0 2px;
background-color: #3498db;
border-radius: 50%;
animation: typingAnimation 1.4s infinite ease-in-out;
}

.typing-dot:nth-child(2) { animation-delay: 0.2s; }
.typing-dot:nth-child(3) { animation-delay: 0.4s; }

@keyframes typingAnimation {
0%, 60%, 100% { transform: translateY(0); }
30% { transform: translateY(-6px); }
}
```

## APPENDIX

```
@keyframes fadeIn {
  from { opacity: 0; transform: translateY(10px); }
  to { opacity: 1; transform: translateY(0); }
}

.chat-box::-webkit-scrollbar {
  width: 6px;
}

.chat-box::-webkit-scrollbar-track {
  background: #f1f1f1;
  border-radius: 3px;
}

.chat-box::-webkit-scrollbar-thumb {
  background: #c1c1c1;
  border-radius: 3px;
}

.chat-box::-webkit-scrollbar-thumb:hover {
  background: #a8a8a8;
}

.system-message {
  text-align: center;
  margin: 10px 0;
  padding: 20px;
  background: rgba(52, 152, 219, 0.1);
  border-radius: 12px;
  color: #2c3e50;
  font-size: 16px;
  font-weight: 600;
  line-height: 1.5;
  border: 1px solid rgba(52, 152, 219, 0.2);
  box-shadow: 0 2px 10px rgba(52, 152, 219, 0.05);
}

.system-message .wave {
  display: inline-block;
  font-size: 24px;
  animation: wave 1.5s infinite;
  transform-origin: 70% 70%;
}

@keyframes wave {
  0% { transform: rotate(0deg); }
  10% { transform: rotate(14deg); }
  20% { transform: rotate(-8deg); }
  30% { transform: rotate(14deg); }
  40% { transform: rotate(-4deg); }
  50% { transform: rotate(10deg); }
  60% { transform: rotate(0deg); }
  100% { transform: rotate(0deg); }
}

.input-container {
  display: flex;
  gap: 1rem;
  align-items: center;
```

```

}

.message-input {
  flex: 1;
  padding: 1rem;
  border: 2px solid #e0e0e0;
  border-radius: var(--border-radius);
  font-size: 1rem;
  transition: all 0.3s ease;
}

.message-input:focus {
  outline: none;
  border-color: var(--secondary-color);
  box-shadow: 0 0 3px rgba(52, 152, 219, 0.1);
}

.send-button {
  color: white;
  border: none;
  border-radius: var(--border-radius);
  cursor: pointer;
  transition: all 0.3s ease;
  font-weight: 600;
  composes: button-base;
  padding: 12px 24px;
  font-size: 16px;
  background: #2C3E50;
  box-shadow: 0 2px 6px rgba(0, 0, 0, 0.1);
}

.send-button:hover {
  background-color: var(--primary-color);
  transform: translateY(-2px);
}

.send-button:active {
  background: #34495E;
  transform: translateY(-2px);
}

::-webkit-scrollbar {
  width: 8px;
}

::-webkit-scrollbar-track {
  background: #f1f1f1;
  border-radius: 4px;
}

::-webkit-scrollbar-thumb {
  background: var(--secondary-color);
  border-radius: 4px;
}

::-webkit-scrollbar-thumb:hover {
  background: var(--primary-color);
}

```

## APPENDIX

```
.test-page-button {
  position: absolute;
  top: 20px;
  left: 20px;
  color: white;
  border: none;
  border-radius: 8px;
  cursor: pointer;
  font-size: 14px;
  text-decoration: none;
  display: flex;
  align-items: center;
  gap: 8px;
  transition: all 0.3s ease;
  z-index: 100;
  composes: button-base;
  padding: 10px 20px;
  background: #2C3E50;
  box-shadow: 0 2px 6px rgba(0, 0, 0, 0.1);
}

.test-page-button:hover {
  background: #34495E;
  transform: translateY(-2px);
  box-shadow: 0 4px 12px rgba(0, 0, 0, 0.3);
}

.test-page-button:active {
  transform: translateY(1px);
  box-shadow: 0 2px 5px rgba(0, 0, 0, 0.3);
}

.upload-message {
  color: #d9534f;
  background-color: #f2dede;
  border: 1px solid #ebccd1;
  border-radius: 4px;
  padding: 10px;
  margin-top: 10px;
  font-size: 14px;
  display: flex;
  align-items: center;
  gap: 5px;
}

.upload-message::before {
  content: " ⚠ ";
  font-size: 18px;
}

#uploadButton {
  color: white;
  border: none;
  padding: 12px 24px;
  border-radius: 6px;
  font-size: 16px;
  font-weight: 600;
  cursor: pointer;
  transition: background-color 0.3s ease, transform 0.2s ease;
```



## APPENDIX

```
    box-shadow: 0 4px 10px rgba(0, 0, 0, 0.1);
    composes: button-base;
    background: green;
    box-shadow: 0 2px 6px rgba(0, 0, 0, 0.1);
  }

#uploadButton:active:not(:disabled) {
  transform: translateY(1px);
  box-shadow: 0 2px 5px rgba(0, 0, 0, 0.2);
}

#uploadButton:hover:not(:disabled) {
  background: green;
  transform: translateY(-2px);
}

#uploadButton:disabled {
  background: #95A5A6;
  cursor: not-allowed;
  opacity: 0.7;
}

.suggestions-container {
  display: flex;
  flex-wrap: wrap;
  margin-top: 10px;
}

.suggestion-bubble {
  background-color: #ffffff;
  color: var(--primary-color);
  border-radius: 20px;
  padding: 10px 15px;
  margin: 5px;
  cursor: pointer;
  transition: background-color 0.3s ease, transform 0.2s ease;
  box-shadow: 0 2px 5px rgba(0, 0, 0, 0.1);
  border: 1px solid #e0e0e0;
}

.suggestion-bubble:hover {
  background-color: #f0f4ff;
  transform: translateY(-2px);
  box-shadow: 0 4px 10px rgba(0, 0, 0, 0.2);
}

.suggestion-bubble:active {
  transform: translateY(1px);
  box-shadow: 0 2px 5px rgba(0, 0, 0, 0.2);
}

.logout-button {
  position: absolute;
  top: 20px;
  right: 20px;
  background-color: #e74c3c;
  color: white;
  border: none;
  padding: 10px 20px;
```

## APPENDIX

```
border-radius: 8px;
cursor: pointer;
font-size: 14px;
font-weight: 600;
text-decoration: none;
display: flex;
align-items: center;
gap: 8px;
transition: all 0.3s ease;
box-shadow: 0 2px 8px rgba(231, 76, 60, 0.2);
z-index: 100;
}

.logout-button:hover {
background-color: #c0392b;
transform: translateY(-2px);
box-shadow: 0 4px 12px rgba(231, 76, 60, 0.3);
}

.logout-button:active {
transform: translateY(1px);
box-shadow: 0 2px 4px rgba(231, 76, 60, 0.2);
}

.user-info {
position: absolute;
top: 20px;
right: 140px;
color: var(--primary-color);
font-size: 14px;
font-weight: 500;
display: flex;
align-items: center;
gap: 5px;
background: rgba(255, 255, 255, 0.9);
padding: 8px 15px;
border-radius: 6px;
box-shadow: 0 2px 8px rgba(0, 0, 0, 0.1);
}

.user-info-icon {
width: 16px;
height: 16px;
opacity: 0.8;
color: var(--primary-color);
}

@media (max-width: 768px) {
.logout-button {
padding: 8px 16px;
font-size: 13px;
}

.user-info {
right: 120px;
font-size: 13px;
padding: 6px 12px;
}
}
```

## APPENDIX

```
.button-base {
  border-radius: 8px;
  cursor: pointer;
  font-weight: 600;
  transition: all 0.3s ease;
  border: none;
  color: white;
  display: flex;
  align-items: center;
  gap: 8px;
}

.logout-button {
  position: absolute;
  top: 20px;
  right: 20px;
  background: #E74C3C;
  color: white;
  border: none;
  padding: 10px 20px;
  border-radius: 8px;
  cursor: pointer;
  font-size: 14px;
  font-weight: 600;
  text-decoration: none;
  display: flex;
  align-items: center;
  gap: 8px;
  transition: all 0.3s ease;
  box-shadow: 0 2px 6px rgba(0, 0, 0, 0.1);
  z-index: 100;
}

.logout-button:hover {
  background: #C0392B;
  transform: translateY(-2px);
}

.button-base:active,
.logout-button:active {
  transform: translateY(1px);
  box-shadow: 0 1px 3px rgba(0, 0, 0, 0.1);
}

.user-dropdown {
  position: absolute;
  top: 20px;
  right: 20px;
  z-index: 100;
}

.user-dropdown-button {
  background: rgba(255, 255, 255, 0.9);
  color: var(--primary-color);
  padding: 8px 15px;
  border-radius: 6px;
  cursor: pointer;
  display: flex;
```

## APPENDIX

```
    align-items: center;
    gap: 8px;
    font-size: 14px;
    font-weight: 500;
    border: 1px solid #e0e0e0;
    box-shadow: 0 2px 8px rgba(0, 0, 0, 0.1);
    transition: all 0.3s ease;
}

.user-dropdown-button:hover {
    background: #f8f9fa;
    transform: translateY(-2px);
    box-shadow: 0 4px 12px rgba(0, 0, 0, 0.15);
}

.user-dropdown-content {
    display: none;
    position: absolute;
    right: 0;
    top: 100%;
    margin-top: 8px;
    background: white;
    border-radius: 6px;
    box-shadow: 0 4px 12px rgba(0, 0, 0, 0.15);
    min-width: 160px;
    overflow: hidden;
}

.user-dropdown-content.show {
    display: block;
    animation: dropdownFadeIn 0.3s ease;
}

@keyframes dropdownFadeIn {
    from { opacity: 0; transform: translateY(-10px); }
    to { opacity: 1; transform: translateY(0); }
}

.user-dropdown-content a {
    color: var(--primary-color);
    padding: 12px 16px;
    text-decoration: none;
    display: flex;
    align-items: center;
    gap: 8px;
    transition: all 0.2s ease;
}

.user-dropdown-content a:hover {
    background: #f8f9fa;
}

.user-dropdown-content .logout-option {
    border-top: 1px solid #e0e0e0;
    color: #e74c3c;
}
</style>
<script type="text/javascript">
    function showInstallConfirmation(tool) {
```

```

const modal = document.getElementById('installModal');
const confirmBtn = document.getElementById('confirmInstall');
const modalText = document.getElementById('modalText');
modal.style.display = 'block';
modalText.textContent = 'Are you sure you want to install ' + tool + '?';

confirmBtn.onclick = function() {
    installTool(tool);
};
}

function cancelInstall() {
    document.getElementById('installModal').style.display = 'none';
}

function installTool(tool) {
    alert('Installation of ' + tool + ' has started. Please wait...');
    document.getElementById('installModal').style.display = 'none';

    const securityButton = document.querySelector(`.security-button[onclick*="${tool}"]`);
    console.log('Found button:', securityButton);
    if (securityButton) {
        securityButton.style.display = 'none';
    } else {
        console.log('Button not found for tool:', tool);
    }

    const loadingDiv = document.createElement('div');
    loadingDiv.id = 'loadingIndicator';
    loadingDiv.innerHTML = `
        <div class="loading-container">
            <div class="loading-content">
                <div class="loading-spinner"></div>
                <div class="loading-text">
                    <h4>Installing ${tool}</h4>
                    <p>Please wait while we set up your security components...</p>
                </div>
            </div>
            <div class="loading-progress">
                <div class="progress-bar"></div>
            </div>
        </div>
    `;

    const styleSheet = document.createElement("style");
    styleSheet.textContent = `
        .loading-container {
            background: linear-gradient(to right, #ffffff, #f8f9fa);
            border-radius: 12px;
            box-shadow: 0 4px 15px rgba(0, 0, 0, 0.1);
            padding: 20px;
            margin: 15px 0;
            border: 1px solid #e9ecef;
        }

        .loading-content {
            display: flex;
            align-items: center;
            gap: 20px;
    `;

```

## APPENDIX

```
        margin-bottom: 15px;
    }

.loading-spinner {
    width: 40px;
    height: 40px;
    border: 4px solid #e9ecef;
    border-top: 4px solid #007bff;
    border-radius: 50%;
    animation: spin 1s linear infinite;
}

.loading-text h4 {
    margin: 0;
    color: #2c3e50;
    font-size: 18px;
    font-weight: 600;
}

.loading-text p {
    margin: 5px 0 0 0;
    color: #6c757d;
    font-size: 14px;
}

.loading-progress {
    background-color: #e9ecef;
    border-radius: 10px;
    height: 6px;
    overflow: hidden;
}

.progress-bar {
    width: 0%;
    height: 100%;
    background: linear-gradient(90deg, #007bff, #00d2ff);
    border-radius: 10px;
    animation: progress 2s ease-in-out infinite;
}

@keyframes spin {
    0% { transform: rotate(0deg); }
    100% { transform: rotate(360deg); }
}

@keyframes progress {
    0% {
        width: 0%;
        opacity: 1;
    }
    50% {
        width: 100%;
        opacity: 0.5;
    }
    100% {
        width: 0%;
        opacity: 1;
    }
}
```

```

    @media (max-width: 480px) {
      .loading-content {
        flex-direction: column;
        text-align: center;
      }
      .loading-text {
        text-align: center;
      }
    }
  `;

  document.head.appendChild(styleSheet);

  const button = document.querySelector(`button[onclick*="${tool}"]`);
  if (button) {
    button.parentNode.insertBefore/loadingDiv, button.nextSibling);
  }

  fetch('install_security.php', {
    method: 'POST',
    headers: {
      'Content-Type': 'application/x-www-form-urlencoded',
    },
    body: 'tool=' + encodeURIComponent(tool)
  })
  .then(response => response.json())
  .then(data => {
    console.log('Installation response:', data);

    const loadingIndicator = document.getElementById('loadingIndicator');
    if (loadingIndicator) {
      loadingIndicator.remove();
    }

    if (data.success) {
      let message = `Installation Status: ${data.status}\n\n`;
      if (data.details) {
        message += `Details:\n${data.details}`;
      }
      alert(message);

      const statusDiv = document.createElement('div');
      statusDiv.style.margin = '10px';
      statusDiv.style.padding = '10px';
      statusDiv.style.border = '1px solid #ccc';
      statusDiv.style.backgroundColor = '#f9f9f9';
      statusDiv.style.wordWrap = 'break-word';
      statusDiv.style.whiteSpace = 'pre-wrap';
      statusDiv.style.maxWidth = '100%';
      statusDiv.style.overflowWrap = 'break-word';
      statusDiv.innerHTML = `<strong>${tool.toUpperCase()} Status:</strong>
${data.status}<br><pre style="white-space: pre-wrap; word-wrap: break-word;">${data.details ||
""}</pre>`;

      if (button) {
        button.parentNode.insertBefore(statusDiv, button.nextSibling);
      }
    } else {

```

```

        alert('Installation failed: ' + data.message);
    }
})
.catch(error => {
    console.error('Error:', error);
    const loadingIndicator = document.getElementById('loadingIndicator');
    if (loadingIndicator) {
        loadingIndicator.remove();
    }
    alert('Error: ' + error);
});
}

function toggleDropdown() {
    const dropdown = document.getElementById('userDropdown');
    dropdown.classList.toggle('show');
    document.addEventListener('click', function(event) {
        const isClickInside = event.target.closest('.user-dropdown');
        if (!isClickInside && dropdown.classList.contains('show')) {
            dropdown.classList.remove('show');
        }
    });
}

function showSettings() {
    window.location.href = 'settings.php';
}
</script>
</head>
<body>
    <div class="chat-container">
        <!-- Admin button -->
        <a href="testing.php" class="test-page-button">
            <svg style="width:16px;height:16px" viewBox="0 0 24 24">
                <path fill="currentColor" d="M19.43 12.98c.04-.32.07-.64.07-.98s-.03-.66-.07-.98l2.11-1.65c.19-.15.24-.42.12-.64l-2-3.46c-.12-.22-.39-.3-.61-.22l-2.49 1c-.52-.4-1.08-.73-1.69-.98l-.38-2.65C14.46 2.18 14.25 2 14 2h-4c-.25 0-.46.18-.49.42l-.38 2.65c-.61.25-1.17.59-1.69.98l-2.49 1c-.23-.09-.49 0-.61.22l-2 3.46c.12.22.39.3.61.22l2.11 1.65c-.04.32-.07.65-.07.98s.03.66.07.98l2.11 1.65c.19-.15.24-.42.12-.64l-2-3.46c-.12-.22-.39-.3-.61-.22l-2.49 1c-.52-.4-1.08-.73-1.69-.98l-.38-2.65c-.61.25-1.17.59-1.69.98l-2.49 1c-.23-.09-.49 0-.61-.22l-2 3.46c.12.22.39.3.61-.22l-2.11-1.65zM12 15.5c-1.93 0-3.5-1.57-3.5-3.5s1.57-3.5 3.5-3.5 3.5-1.57 3.5-3.5 1.57-3.5 3.5-3.5 3.5-1.57 3.5-3.5 3.5z"/>
            </svg>
            Admin
        </a>

        <!-- User dropdown -->
        <div class="user-dropdown">
            <div class="user-dropdown-button" onclick="toggleDropdown()">
                <svg class="user-info-icon" viewBox="0 0 24 24">
                    <path fill="currentColor" d="M12,4A4,4 0 0,1 16,8A4,4 0 0,1 12,12A4,4 0 0,1 8,8A4,4 0 0,1 12,4M12,14C16.42,14 20,15.79 20,18V20H4V18C4,15.79 7.58,14 12,14Z" />
                </svg>
                <?php echo htmlspecialchars($_SESSION['username']); ?>
                <svg style="width:16px;height:16px" viewBox="0 0 24 24">
                    <path fill="currentColor" d="M7,10L12,15L17,10H7Z" />
                </svg>
            </div>
            <div class="user-dropdown-content" id="userDropdown">

```



```

    <a href="#" onclick="showSettings()">
      <svg style="width:16px;height:16px" viewBox="0 0 24 24">
        <path fill="currentColor" d="M12,15.5A3.5,3.5 0 0,1 8.5,12A3.5,3.5 0 0,1
12,8.5A3.5,3.5 0 0,1 15.5,12A3.5,3.5 0 0,1 12,15.5M19.43,12.97C19.47,12.65 19.5,12.33
19.5,12C19.5,11.67 19.47,11.34 19.43,11L21.54,9.37C21.73,9.22 21.78,8.95
21.66,8.73L19.66,5.27C19.54,5.05 19.27,4.96 19.05,5.05L16.56,6.05C16.04,5.66 15.5,5.32
14.87,5.07L14.5,2.42C14.46,2.18 14.25,2 14,2H10C9.75,2 9.54,2.18 9.5,2.42L9.13,5.07C8.5,5.32
7.96,5.66 7.44,6.05L4.95,5.05C4.73,4.96 4.46,5.05 4.34,5.27L2.34,8.73C2.21,8.95 2.27,9.22
2.46,9.37L4.57,11C4.53,11.34 4.5,11.67 4.5,12C4.5,12.33 4.53,12.65
4.57,12.97L2.46,14.63C2.27,14.78 2.21,15.05 2.34,15.27L4.34,18.73C4.46,18.95 4.73,19.03
4.95,18.95L7.44,17.94C7.96,18.34 8.5,18.68 9.13,18.93L9.5,21.58C9.54,21.82 9.75,22
10,22H14C14.25,22 14.46,21.82 14.5,21.58L14.87,18.93C15.5,18.67 16.04,18.34
16.56,17.94L19.05,18.95C19.27,19.03 19.54,18.95 19.66,18.73L21.66,15.27C21.78,15.05 21.73,14.78
21.54,14.63L19.43,12.97Z" />
      </svg>
      Settings
    </a>
    <a href="#" class="logout-option" onclick="confirmLogout(event)">
      <svg style="width:16px;height:16px" viewBox="0 0 24 24">
        <path fill="currentColor" d="M16,17V14H9V10H16V7L21,12L16,17M14,2A2,2 0 0,1
16,4V6H14V4H5V20H14V18H16V20A2,2 0 0,1 14,22H5A2,2 0 0,1 3,20V4A2,2 0 0,1 5,2H14Z" />
      </svg>
      Logout
    </a>
  </div>
</div>

<h1>
  <svg style="width:24px;height:24px;vertical-align:middle" viewBox="0 0 24 24">
    <path fill="currentColor" d="M12,2A2,2 0 0,1 14,4C14,4.74 13.6,5.39 13,5.73V7H14A7,7 0
0,1 21,14H22A1,1 0 0,1 23,15V18A1,1 0 0,1 22,19H21V20A2,2 0 0,1 19,22H5A2,2 0 0,1
3,20V19H2A1,1 0 0,1 1,18V15A1,1 0 0,1 2,14H3A7,7 0 0,1 10,7H11V5.73C10,4.5,39 10,4.74
10,4A2,2 0 0,1 12,2M7.5,13A2.5,2.5 0 0,0 5,15.5A2.5,2.5 0 0,0 7.5,18A2.5,2.5 0 0,0 10,15.5A2.5,2.5 0
0,0 7.5,13M16.5,13A2.5,2.5 0 0,0 14,15.5A2.5,2.5 0 0,0 16.5,18A2.5,2.5 0 0,0 19,15.5A2.5,2.5 0 0,0
16.5,13Z" />
  </svg>
  Multi-Fuzzer AI Chatbot System
</h1>

<div class="file-upload">
  <h3> 🛡️ Security Verification</h3>
  <form action="index1.php" method="post" enctype="multipart/form-data" onsubmit="return
confirmUpload()">
    <input type="file" name="security_file" id="security_file" onchange="checkFileUpload()">
    <input type="submit" value="Upload & Start Fuzzing" name="submit" id="uploadButton"
disabled>
  </form>
</div>

<div class="chat-box" id="chatBox">
  <div class="message bot">
    <div class="avatar"> 🤖 </div>
    <div class="message-content" style="text-align: left;">
      <span class="wave"> 🌊 </span>
      <strong>Hello! I'm your AI assistant.</strong><br>
      Please upload your security verification document to begin our conversation.
    </div>
  </div>
</div>

```

```

<div class="input-container">
  <input type="text" class="message-input" id="messageInput" placeholder="Type your message
here...">
  <button class="send-button" onclick="sendMessage()">
    <svg style="width:16px;height:16px" viewBox="0 0 24 24">
      <path fill="currentColor" d="M2,21L23,12L2,3V10L17,12L2,14V21Z" />
    </svg>
    Send
  </button>
</div>

<div id="suggestions" class="suggestions-container"></div>
</div>

<script>
function checkFileUpload() {
  const fileInput = document.getElementById('security_file');
  const messageInput = document.getElementById('messageInput');
  const sendButton = document.querySelector('.send-button');
  const uploadButton = document.getElementById('uploadButton');
  const uploadMessage = document.getElementById('uploadMessage');

  if (fileInput.files.length > 0) {
    uploadButton.disabled = false;
  } else {
    uploadButton.disabled = true;
  }
}

function confirmUpload() {
  const fileInput = document.getElementById('security_file');
  if (fileInput.files.length === 0) {
    alert("Please select a file to upload.");
    return false;
  }
  return confirm("Are you sure you want to upload this file and start fuzzing?");
}

function sendMessage() {
  const messageInput = document.getElementById('messageInput');
  const message = messageInput.value.trim();

  if (message === "") return;

  const chatBox = document.getElementById('chatBox');
  const timestamp = new Date().toLocaleTimeString();

  chatBox.innerHTML += `
    <div class="message user">
      <div class="message-content">
        ${message}
      <div class="timestamp">${timestamp}</div>
    </div>
    <div class="avatar">${'<?php echo strtoupper(substr($_SESSION["username"], 0,
1)); ?>'}</div>
  </div>
  `;

```

```

chatBox.innerHTML += `
  <div class="typing-indicator" id="typing-indicator">
    <div class="typing-dot"></div>
    <div class="typing-dot"></div>
    <div class="typing-dot"></div>
  </div>
`;
chatBox.scrollTop = chatBox.scrollHeight;

fetch('index.php', {
  method: 'POST',
  headers: {
    'Content-Type': 'application/x-www-form-urlencoded',
  },
  body: `message=${encodeURIComponent(message)}`
})
.then(response => response.json())
.then(data => {
  const typingIndicator = document.getElementById('typing-indicator');
  if (typingIndicator) {
    typingIndicator.remove();
  }

  chatBox.innerHTML += `
    <div class="message bot">
      <div class="avatar">🤖</div>
      <div class="message-content">
        ${data.response}
        <div class="timestamp">${new Date().toLocaleTimeString()}</div>
      </div>
    </div>
  `;
  chatBox.scrollTop = chatBox.scrollHeight;
})
.catch(error => {
  console.error('Error:', error);
  const typingIndicator = document.getElementById('typing-indicator');
  if (typingIndicator) {
    typingIndicator.remove();
  }
  chatBox.innerHTML += `
    <div class="system-message">
      Error: Could not process your request
    </div>
  `;
  chatBox.scrollTop = chatBox.scrollHeight;
});

messageInput.value = "";
}

document.getElementById('messageInput').addEventListener('keypress', function(e) {
  if (e.key === 'Enter') {
    sendMessage();
  }
});

const suggestions = [
  "What vulnerabilities were identified in the crash report?",

```

```

    "Can you provide a detailed analysis of the crash?",
    "What are the potential impacts of the vulnerabilities?",
    "What mitigation strategies can be implemented?",
    "Are there any recommended patches or updates?",
    "How can I prevent similar crashes in the future?",
    "Can you summarize the key findings from the crash report?",
    "What tools can I use to further analyze the vulnerabilities?",
    "Is there a timeline for implementing the mitigation plan?",
    "Can you provide links to resources on these vulnerabilities?"
  ];

function getRandomSuggestions(suggestions, count) {
  const shuffled = suggestions.sort(() => 0.5 - Math.random());
  return shuffled.slice(0, count);
}

function displaySuggestions() {
  const suggestionsContainer = document.getElementById('suggestions');
  suggestionsContainer.innerHTML = "";
  const randomSuggestions = getRandomSuggestions(suggestions, 2);

  randomSuggestions.forEach(suggestion => {
    const bubble = document.createElement('div');
    bubble.className = 'suggestion-bubble';
    bubble.textContent = suggestion;

    bubble.onclick = function() {
      document.getElementById('messageInput').value = suggestion;
    };

    suggestionsContainer.appendChild(bubble);
  });
}

window.onload = function() {
  if (document.referrer.includes('index1.php')) {
    const messageInput = document.getElementById('messageInput');
    messageInput.value = "Please analyze this fuzzing crash report and provide actionable recommendations following CVE format.";
    sendMessage();
  }
  displaySuggestions();
};

function confirmLogout(event) {
  event.preventDefault();
  if (confirm('Are you sure you want to logout?')) {
    window.location.href = 'logout.php';
  }
}
</script>
</body>
</html>

```

**index1.php**

```

<?php
session_start();
if (!isset($_SERVER['PHP_AUTH_USER'])) {
    header('WWW-Authenticate: Basic realm="Command Execute Interface");
    header('HTTP/1.0 401 Unauthorized');
    echo 'Authentication required';
    exit;
}

error_reporting(E_ALL);
ini_set('display_errors', 1);

$output = "";
$error = "";
$auto_command = "";

if (isset($_FILES['security_file'])) {
    $filename = pathinfo($_FILES["security_file"])[ "name" ], PATHINFO_FILENAME);
    $auto_command = "RUN_PARALLEL 30 " . $filename;
    error_log("Command generated for file: " . $filename);
}

if (isset($_GET['command'])) {
    $auto_command = htmlspecialchars($_GET['command']);
}

if ($_SERVER['REQUEST_METHOD'] === 'POST' && isset($_POST['command'])) {
    $command = trim($_POST['command']);

    $command = escapeshellcmd($command);
    $timestamp = date('Y-m-d_H-i-s');

    //Stop Fuzzing Command
    if (strcasecmp($command, 'STOP_FUZZING') === 0) {
        $diagnostic_command = "cd /home/einjun && /home/einjun/stop_fuzzing.sh";
    }
    //Parallel Fuzzing Command
    elseif (preg_match('/^RUN_PARALLEL\s+(\d+)\s+(\w+)\$/i', $command, $matches)) {
        $minutes = (int)$matches[1];
        $target = $matches[2];
        $diagnostic_command = "rm -rf /home/einjun/Desktop/outputhonggfuzz/* && cd /home/einjun
&& /home/einjun/run_parallel.sh {$minutes} {$target}";
    }
    //AFL++ Fuzzing Command
    elseif (preg_match('/^RUN_AFL_TIMED\s+(\d+)\s+(\w+)\$/i', $command, $matches)) {
        $minutes = (int)$matches[1];
        $target = $matches[2];
        $diagnostic_command = "cd /home/einjun && /home/einjun/run_afl.sh {$minutes} {$target}";
    } else {
        $diagnostic_command = "cd /home/einjun && export
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/home/einjun/AFLplusplus && export
HOME=/home/einjun && " . $command;
    }

    $output_array = array();
    $return_status = 0;

```

## APPENDIX

```
$debug_info = "Debug Info:\n";
$debug_info .= "Current PHP User: " . exec('whoami') . "\n";
$debug_info .= "Current PHP Path: " . exec('pwd') . "\n";
$debug_info .= "Executing command: " . $diagnostic_command . "\n";

$sudo_command = "/usr/bin/sudo -n -u einjun /bin/bash -c \"'\" . $diagnostic_command . "\" 2>&1";

exec($sudo_command, $output_array, $return_status);

if ($return_status === 0) {
    $output = $debug_info . "\nCommand Output:\n" . implode("\n", $output_array);
} else {
    $error = $debug_info . "\nCommand execution failed with status: " . $return_status . "\nOutput: " .
    implode("\n", $output_array);
}

echo "<div id='debugInfo' style='background: #1e1e1e; color: #00ff00; padding: 15px; margin: 20px
0; font-family: monospace; border-radius: 5px; max-height: 300px; overflow-y: auto;*>
    <h3 style='color: #fff; margin-top: 0;*>Debug Information</h3>
    <pre id='debugContent'>Monitoring for crashes...</pre>
    </div>";
}

$autoSubmit = isset($_FILES['security_file']);
$showLoading = isset($_FILES['security_file']);
?>

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Ubuntu CLI Web Interface</title>
    <style>
        :root {
            --primary-color: #2c3e50;
            --secondary-color: #3498db;
            --accent-color: #e74c3c;
            --background-color: #1e1e1e;
            --chat-bg: #2d2d2d;
            --text-color: #ffffff;
            --border-radius: 12px;
        }

        body {
            font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
            margin: 0;
            padding: 20px;
            background: var(--background-color);
            color: var(--text-color);
            min-height: 100vh;
        }

        .container {
            max-width: 1200px;
            margin: 0 auto;
        }

        h1 {
```

## APPENDIX

```
    color: var(--text-color);
    margin-bottom: 1.5rem;
    font-size: 1.8rem;
    font-weight: 600;
  }

.terminal {
  background: #000;
  padding: 20px;
  border-radius: var(--border-radius);
  margin: 20px 0;
  white-space: pre-wrap;
  font-family: monospace;
  box-shadow: 0 4px 6px rgba(0, 0, 0, 0.1);
}

input[type="text"] {
  width: 100%;
  padding: 12px;
  margin: 10px 0;
  background: var(--chat-bg);
  border: 1px solid #444;
  color: var(--text-color);
  border-radius: var(--border-radius);
  font-size: 1rem;
}

input[type="submit"] {
  padding: 12px 24px;
  background: var(--secondary-color);
  border: none;
  color: white;
  cursor: pointer;
  border-radius: var(--border-radius);
  font-weight: 600;
  transition: all 0.3s ease;
}

input[type="submit"]:hover {
  background: var(--primary-color);
  transform: translateY(-2px);
}

.error {
  color: var(--accent-color);
}

.help-text {
  background: var(--chat-bg);
  padding: 20px;
  border-radius: var(--border-radius);
  margin: 20px 0;
  box-shadow: 0 4px 6px rgba(0, 0, 0, 0.1);
}

.help-text h3 {
  color: var(--secondary-color);
  margin-top: 0;
}
```

## APPENDIX

```
code {
  display: inline-block;
  background: rgba(0, 0, 0, 0.2);
  padding: 4px 8px;
  border-radius: 4px;
  margin: 4px 0;
  font-family: monospace;
}

::-webkit-scrollbar {
  width: 8px;
}

::-webkit-scrollbar-track {
  background: var(--chat-bg);
  border-radius: 4px;
}

::-webkit-scrollbar-thumb {
  background: var(--secondary-color);
  border-radius: 4px;
}

::-webkit-scrollbar-thumb:hover {
  background: var(--primary-color);
}

.command-form {
  display: flex;
  gap: 10px;
  margin: 20px 0;
}

.command-form input[type="text"] {
  flex: 1;
  margin: 0;
}

@media (max-width: 768px) {
  .command-form {
    flex-direction: column;
  }

  .command-form input[type="submit"] {
    width: 100%;
  }
}

.loading-overlay {
  position: fixed;
  top: 0;
  left: 0;
  width: 100%;
  height: 100%;
  background: rgba(0, 0, 0, 0.85);
  display: flex;
  justify-content: center;
  align-items: center;
}
```



## APPENDIX

```
        z-index: 9999999;
    }

    .loading-status {
        background: #2d2d2d;
        color: #00ff00;
        padding: 30px 50px;
        border-radius: 10px;
        text-align: center;
        box-shadow: 0 0 20px rgba(0, 255, 0, 0.2);
        max-width: 80%;
        z-index: 999999;
    }

    .loading-spinner {
        display: inline-block;
        width: 50px;
        height: 50px;
        border: 4px solid #00ff00;
        border-radius: 50%;
        border-top-color: transparent;
        animation: spin 1s linear infinite;
        margin-bottom: 20px;
        z-index: 999999;
    }

    @keyframes spin {
        to {transform: rotate(360deg);}
    }

    .status-text {
        font-size: 1.2em;
        margin-bottom: 10px;
        z-index: 999999;
    }

    }

    .time-remaining {
        color: #888;
        font-size: 1em;
        margin-top: 10px;
        z-index: 999999;
    }

    }

    #debugInfo {
        position: fixed;
        bottom: 20px;
        left: 20px;
        right: 20px;
        max-height: 200px;
        overflow-y: auto;
        background: rgba(0, 0, 0, 0.8);
        z-index: 1001;
    }
</style>

</head>
<body>
```

```

<div class="container">
  <h1>  Multi-Fuzzer CLI Web Interface</h1>

  <div class="help-text">
    <h3>  Usage Examples:</h3>
    <p>Run AFL++ for specific duration and target:</p>
    <code>RUN_AFL_TIMED 30 testsleep</code><br>
    <code>RUN_AFL_TIMED 30 test1</code><br>
    <code>RUN_AFL_TIMED 30 testwake</code>
    <p>Run AFL++ and Honggfuzz in parallel:</p>
    <code>RUN_PARALLEL 30 testsleep</code><br>
    <code>RUN_PARALLEL 30 test1</code>
    <p>Stop all fuzzing processes:</p>
    <code>STOP_FUZZING</code>
    <p>Regular commands:</p>
    <code>ls</code><br>
    <code>pwd</code>
  </div>

  <form method="POST" class="command-form">
    <input type="text" name="command" placeholder="Enter your command" value="<?php echo
htmlspecialchars($auto_command); ?>" required>
    <input type="submit" value="Execute">
  </form>

  <?php if ($showLoading): ?>
  <div class="loading-overlay">
    <div class="loading-status">
      <div class="loading-spinner"></div>
      <div class="status-text">
        <span id="statusMessage">Fuzzing in progress...</span>
        <div class="time-remaining">
          Time remaining: <span id="timeRemaining">30:00</span>
        </div>
      </div>
    </div>
  </div>
</div>

<div id="debugInfo">
  <h3 style="color: #fff; margin-top: 0;">Debug Information</h3>
  <pre id="debugContent">Monitoring for crashes...</pre>
</div>
<?php endif; ?>

<?php if ($error): ?>
  <div class="terminal error">
    <?php echo htmlspecialchars($error); ?>
  </div>
<?php endif; ?>

<?php if ($output): ?>
  <div class="terminal">
    <?php echo htmlspecialchars($output); ?>
  </div>
<?php endif; ?>
</div>

<script>
  function startTimer(duration) {

```

```

let timer = duration;
const timerDisplay = document.getElementById('timeRemaining');

const countdown = setInterval(function () {
    const minutes = parseInt(timer / 60, 10);
    const seconds = parseInt(timer % 60, 10);

    timerDisplay.textContent = minutes.toString().padStart(2, '0') + ':' +
        seconds.toString().padStart(2, '0');

    if (--timer < 0) {
        clearInterval(countdown);
        document.getElementById('statusMessage').textContent = 'Fuzzing completed!';
        timerDisplay.textContent = '00:00';
    }
}, 1000);
}

window.onload = function() {
    <?php if ($autoSubmit): ?>
        document.querySelector('.command-form').submit();
    <?php endif; ?>
    startTimer(30 * 60); // 30 minutes
    checkCrashFile();
}

let checkCount = 0;

function ensureDebugElementsExist() {
    let debugInfo = document.getElementById('debugInfo');
    if (!debugInfo) {
        debugInfo = document.createElement('div');
        debugInfo.id = 'debugInfo';
        debugInfo.style.cssText = 'background: #1e1e1e; color: #00ff00; padding: 15px; margin:
20px 0; font-family: monospace; border-radius: 5px; max-height: 300px; overflow-y: auto; position:
fixed; bottom: 20px; left: 20px; right: 20px; z-index: 1001;';
        debugInfo.innerHTML = `
            <h3 style="color: #fff; margin-top: 0;">Debug Information</h3>
            <pre id="debugContent">Initializing debug display...</pre>
        `;
        document.body.appendChild(debugInfo);
    }
    return document.getElementById('debugContent');
}

function updateDebugDisplay(info) {
    const debugContent = ensureDebugElementsExist();
    if (!debugContent) {
        console.error('Failed to create or find debug elements');
        return;
    }

    const timestamp = new Date().toLocaleTimeString();
    let debugText = `[${timestamp}] Check #${checkCount}\n`;

    if (info.error) {
        debugText += 'Error occurred:\n' +
            JSON.stringify(info, null, 2);
    } else {

```

```

        debugText += 'Raw Response Data:\n' +
            JSON.stringify(info, null, 2) + '\n\n' +
            'crash_found value: ' + info.crash_found + '\n' +
            'typeof crash_found: ' + typeof info.crash_found;
    }

    debugContent.innerHTML = debugText;
    debugContent.scrollTop = debugContent.scrollHeight;
}

function checkCrashFile() {
    checkCount++;
    ensureDebugElementsExist();

    fetch('check_crash.php')
        .then(response => response.json())
        .then(data => {
            updateDebugDisplay(data);

            if (data.crash_found) {
                const statusMessage = document.getElementById('statusMessage');
                const loadingOverlay = document.querySelector('.loading-overlay');

                if (statusMessage && loadingOverlay) {
                    loadingOverlay.style.display = 'flex';
                    statusMessage.textContent = 'Crash found! Redirecting...';

                    setTimeout(() => {
                        window.location.href = 'index.php';
                    }, 2000);
                } else {
                    const overlay = document.createElement('div');
                    overlay.className = 'loading-overlay';
                    overlay.style.cssText = 'display: flex; position: fixed; top: 0; left: 0; width: 100%; height: 100%; background: rgba(0, 0, 0, 0.8); z-index: 1002; justify-content: center; align-items: center;';

                    const status = document.createElement('div');
                    status.className = 'loading-status';
                    status.innerHTML = `
                        <div class="loading-spinner"></div>
                        <div class="status-text">
                            <span id="statusMessage">Crash found! Redirecting...</span>
                        </div>
                    `;

                    overlay.appendChild(status);
                    document.body.appendChild(overlay);

                    setTimeout(() => {
                        window.location.href = 'index.php';
                    }, 2000);
                }
            } else {
                if (checkCount < 1800) {
                    setTimeout(checkCrashFile, 1000);
                }
            }
        })
}

```

## APPENDIX

```
.catch(error => {  
  updateDebugDisplay({  
    'error': error.message,  
    'check_number': checkCount,  
    'stack': error.stack  
  });  
  if (checkCount < 1800) {  
    setTimeout(checkCrashFile, 1000);  
  }  
});  
}  
</script>  
</body>  
</html>
```

**install\_security.php**

```

<?php
header('Content-Type: application/json');
error_reporting(E_ALL);
ini_set('display_errors', 1);
error_log("Installation process started");

$tool = isset($_POST['tool']) ? $_POST['tool'] : "";
error_log("Tool requested: " . $tool);

$allowed_tools = [
    'apparmor',
    'grsecurity',
    'pax',
    'asan',
    'hardened_malloc',
    'exploit_mitigations',
    'disk_flooding_attack_mitigation',
    'inodes_exhaustion_mitigation'
];

if (!in_array($tool, $allowed_tools)) {
    error_log("Invalid tool specified: " . $tool);
    echo json_encode(['success' => false, 'message' => 'Invalid tool specified']);
    exit;
}

// Map tools to their installation scripts
$script_map = [
    'apparmor' => 'install_apparmor.sh',
    'grsecurity' => 'install_grsecurity.sh',
    'pax' => 'install_pax.sh',
    'asan' => 'install_asan.sh',
    'hardened_malloc' => 'install_hardened_malloc.sh',
    'exploit_mitigations' => 'install_exploit_mitigations.sh',
    'disk_flooding_attack_mitigation' => 'mitigate_disk_flood_attack.sh',
    'inodes_exhaustion_mitigation' => 'mitigate_inodes_exhaustion.sh'
];

$script_path = "/home/einjun/scripts/" . $script_map[$tool];
error_log("Script path: " . $script_path);

if (!file_exists($script_path)) {
    error_log("Script not found at: " . $script_path);
    echo json_encode(['success' => false, 'message' => 'Installation script not found']);
    exit;
}

// Create a temporary expect script to handle the sudo password
$expect_script = <<<EOT
#!/usr/bin/expect -f
log_file /tmp/expect_log.txt
exp_internal 1
set timeout -1
spawn /usr/bin/sudo /bin/bash $script_path
expect {
    "*password*" {
        send "\r"
    }
}
EOT

```

## APPENDIX

```
        exp_continue
    }
    eof
}
EOT;

$expect_path = "/tmp/install_script_" . time() . ".exp";
file_put_contents($expect_path, $expect_script);
chmod($expect_path, 0700);
error_log("Expect script created at: " . $expect_path);

// Run the expect script
$command = "/usr/bin/expect " . escapeshellarg($expect_path) . " 2>&1";
error_log("Executing command: " . $command);

$output = [];
$return_var = 0;

exec($command, $output, $return_var);
error_log("Command output: " . print_r($output, true));
error_log("Return value: " . $return_var);
unlink($expect_path);

// Check installation status
$status_command = "";
switch($tool) {
    case 'asan':
        $status_command = "dpkg -l | grep -E 'clang|llvm' || which clang";
        break;
    case 'apparmor':
        $status_command = "systemctl is-active apparmor && echo 'AppArmor is active and running' || echo 'AppArmor is not running'";
        break;
    case 'hardened_malloc':
        $status_command = "ls /usr/lib/libhardened_malloc.so";
        break;
    case 'exploit_mitigations':
        $status_command = "sysctl kernel.randomize_va_space";
        break;
    case 'disk_flooding_attack_mitigation':
        $status_command = "df -h /tmp";
        break;
    case 'inodes_exhaustion_mitigation':
        $status_command = "stat -f /tmp";
        break;
}

$status_output = [];
$status_return = 0;
if ($status_command) {
    exec($status_command, $status_output, $status_return);
}

// Special handling for mitigate_buffer_overflow and inodes_exhaustion_mitigation
if ($tool === 'disk_flooding_attack_mitigation') {
    $command = "/bin/bash " . escapeshellarg($script_path) . " 2>&1";
    error_log("Executing direct command: " . $command);

    $output = [];
```

## APPENDIX

```
$return_var = 0;

exec($command, $output, $return_var);
error_log("Command output: " . print_r($output, true));
error_log("Return value: " . $return_var);

$file_exists = file_exists('/tmp/fuzz_test');
if (!$file_exists) {
    echo json_encode([
        'success' => true,
        'message' => 'Buffer overflow mitigation completed',
        'status' => 'Successfully removed vulnerable file',
        'details' => implode("\n", $output)
    ]);
} else {
    if (is_writable('/tmp/fuzz_test')) {
        unlink('/tmp/fuzz_test');
        echo json_encode([
            'success' => true,
            'message' => 'Buffer overflow mitigation completed using PHP',
            'status' => 'Successfully removed vulnerable file',
            'details' => implode("\n", $output)
        ]);
    } else {
        echo json_encode([
            'success' => false,
            'message' => 'Disk flooding attack mitigation failed: File could not be removed',
            'details' => implode("\n", $output)
        ]);
    }
}
exit;
} else if ($tool === 'inodes_exhaustion_mitigation') {
    try {
        $target_dir = '/tmp/inode_flood/';
        $before_stat = [];
        exec("stat -f /tmp", $before_stat);

        $cleanup_command = "/usr/bin/sudo rm -rf " . escapeshellarg($target_dir) . " 2>&1";
        exec($cleanup_command, $cleanup_output, $cleanup_return);

        $command = "/usr/bin/sudo /bin/bash " . escapeshellarg($script_path) . " 2>&1";
        exec($command, $script_output, $script_return);

        $after_stat = [];
        exec("stat -f /tmp", $after_stat);

        $output_array = array_merge(
            ["=== INODE EXHAUSTION MITIGATION REPORT ==="],
            ["\n[1] INITIAL SYSTEM STATE"],
            ["-----"],
            $before_stat,
            ["\n[2] CLEANUP OPERATIONS"],
            ["-----"],
            ["Target Directory: " . $target_dir],
            ["Cleanup Command: " . $cleanup_command],
            ["Cleanup Results:"],
            $cleanup_output,
            ["\n[3] FINAL SYSTEM STATE"],
        );
    }
}
```



## APPENDIX

```
["-----"],
$after_stat,
["\n[4] MITIGATION SUMMARY"],
["-----"],
["Status: " . ($cleanup_return === 0 ? "SUCCESS - Inodes cleaned successfully" : "WARNING
- Some issues encountered")],
["Directory Exists: " . (is_dir($target_dir) ? "Yes (Warning)" : "No (Good)"),
["Cleanup Return Code: " . $cleanup_return],
["\n=== END OF REPORT ==="]
);

if (!empty($before_stat) && !empty($after_stat)) {
    $before_inodes = preg_match('/Inodes:.*Free:\s+(\d+)/', implode("\n", $before_stat),
$before_matches) ? $before_matches[1] : null;
    $after_inodes = preg_match('/Inodes:.*Free:\s+(\d+)/', implode("\n", $after_stat),
$after_matches) ? $after_matches[1] : null;

    if ($before_inodes !== null && $after_inodes !== null) {
        $inode_diff = $after_inodes - $before_inodes;
        $output_array = array_merge($output_array, [
            "\n[5] INODE ANALYSIS",
            "-----",
            "Free Inodes Before: " . number_format($before_inodes),
            "Free Inodes After: " . number_format($after_inodes),
            "Inodes Freed: " . ($inode_diff > 0 ? "+" . number_format($inode_diff) :
number_format($inode_diff))
        ]);
    }
}

$clean_output = implode("\n", array_map(function($line) {
    return trim(preg_replace('/^[^x20-x7E\x0A\x0D]/', "", $line));
}, $output_array));

$success = !is_dir($target_dir) || $cleanup_return === 0;

header('Content-Type: application/json');
echo json_encode([
    'success' => $success,
    'message' => $success ? 'Inodes exhaustion mitigation completed' : 'Mitigation completed with
warnings',
    'status' => $success ? 'Successfully cleaned up excessive files' : 'Cleanup process completed
with some issues',
    'details' => $clean_output
], JSON_UNESCAPED_SLASHES);

} catch (Exception $e) {
    header('Content-Type: application/json');
    echo json_encode([
        'success' => false,
        'message' => 'Mitigation failed',
        'details' => 'Error: ' . $e->getMessage()
    ]);
}
exit;
}

if ($tool === 'apparmor' && file_exists('/var/log/apparmor/status.txt')) {
    $apparmor_status = file_get_contents('/var/log/apparmor/status.txt');
```

## APPENDIX

```
$installation_status = (strpos($apparmor_status, 'apparmor module is loaded') !== false)
    ? "Successfully installed and running"
    : "Installation may have failed";
$status_output = [$apparmor_status];
} else {
    $installation_status = $status_return === 0 ? "Successfully installed" : "Installation may have
failed";
}

if ($return_var === 0) {
    error_log("Installation completed successfully");
    echo json_encode([
        'success' => true,
        'message' => 'Installation completed',
        'status' => $installation_status,
        'details' => implode("\n", $status_output)
    ]);
} else {
    error_log("Installation failed");
    echo json_encode([
        'success' => false,
        'message' => 'Installation failed: ' . implode("\n", $output),
        'return_var' => $return_var
    ]);
}
```

**login.php**

```

<?php
session_start();
$error = "";

$db_host = 'localhost';
$db_user = 'root';
$db_pass = "";
$db_name = 'chatbot_db';

$conn = new mysqli($db_host, $db_user, $db_pass, $db_name);

if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}

if ($_SERVER["REQUEST_METHOD"] == "POST") {
    $username = $conn->real_escape_string($_POST['username']);
    $password = $_POST['password'];

    $sql = "SELECT u.id, u.username, u.password, u2fa.secret, u2fa.enabled
            FROM users u
            LEFT JOIN user_2fa u2fa ON u.id = u2fa.user_id
            WHERE u.username = ?";
    $stmt = $conn->prepare($sql);
    $stmt->bind_param("s", $username);
    $stmt->execute();
    $result = $stmt->get_result();

    if ($result->num_rows === 1) {
        $user = $result->fetch_assoc();
        if (password_verify($password, $user['password'])) {
            if ($user['enabled'] == 1) {
                $_SESSION['temp_user_id'] = $user['id'];
                $_SESSION['temp_username'] = $user['username'];
                $_SESSION['requires_2fa'] = true;
                $_SESSION['2fa_secret'] = $user['secret'];

                header("Location: verify_2fa.php");
                exit();
            } else {
                $_SESSION['user_id'] = $user['id'];
                $_SESSION['username'] = $user['username'];
                header("Location: index.php");
                exit();
            }
        } else {
            $error = "Invalid username or password";
        }
    } else {
        $error = "Invalid username or password";
    }
}

$stmt->close();
}
?>

<!DOCTYPE html>

```

## APPENDIX

```
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Login - Multi-Fuzzer AI Chatbot System</title>
  <style>
    :root {
      --primary-color: #2c3e50;
      --secondary-color: #3498db;
      --background-color: #f4f6f9;
      --error-color: #e74c3c;
      --success-color: #2ecc71;
      --text-color: #2c3e50;
      --text-muted: #7f8c8d;
      --border-color: #e0e0e0;
      --shadow-sm: 0 2px 4px rgba(0,0,0,0.05);
      --shadow-md: 0 4px 6px rgba(0,0,0,0.1);
      --shadow-lg: 0 8px 24px rgba(0,0,0,0.1);
      --gradient-primary: linear-gradient(135deg, var(--primary-color), var(--secondary-color));
    }

    * {
      margin: 0;
      padding: 0;
      box-sizing: border-box;
    }

    body {
      font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
      background-color: var(--background-color);
      min-height: 100vh;
      display: flex;
      align-items: center;
      justify-content: center;
      padding: 20px;
      color: var(--text-color);
      line-height: 1.6;
      position: relative;
      overflow: hidden;
    }

    .background-pattern {
      position: fixed;
      top: 0;
      left: 0;
      width: 100%;
      height: 100%;
      background-image:
        radial-gradient(circle at 25% 25%, rgba(52, 152, 219, 0.1) 0%, transparent 50%),
        radial-gradient(circle at 75% 75%, rgba(44, 62, 80, 0.1) 0%, transparent 50%);
      z-index: -1;
    }

    .login-container {
      background: white;
      padding: 2.5rem;
      border-radius: 16px;
      box-shadow: var(--shadow-lg);
      width: 100%;
    }
```

## APPENDIX

```
    max-width: 420px;
    position: relative;
    overflow: hidden;
    backdrop-filter: blur(10px);
    border: 1px solid rgba(255, 255, 255, 0.2);
    animation: slideUp 0.5s ease-out;
  }

  @keyframes slideUp {
    from {
      opacity: 0;
      transform: translateY(20px);
    }
    to {
      opacity: 1;
      transform: translateY(0);
    }
  }

  .login-header {
    text-align: center;
    margin-bottom: 2rem;
    position: relative;
  }

  .login-header h1 {
    color: var(--text-color);
    font-size: 1.8rem;
    margin-bottom: 0.5rem;
    display: flex;
    align-items: center;
    justify-content: center;
    gap: 0.8rem;
    position: relative;
  }

  .login-header svg {
    color: var(--secondary-color);
    filter: drop-shadow(0 2px 4px rgba(52, 152, 219, 0.2));
  }

  .login-header p {
    color: var(--text-muted);
    font-size: 0.95rem;
    margin-top: 0.5rem;
  }

  .form-group {
    margin-bottom: 1.8rem;
    position: relative;
  }

  .form-group label {
    display: block;
    margin-bottom: 0.8rem;
    color: var(--text-color);
    font-weight: 500;
    font-size: 0.95rem;
    transition: color 0.3s ease;
```

## APPENDIX

```
}

.form-group input {
  width: 100%;
  padding: 1rem;
  border: 2px solid var(--border-color);
  border-radius: 12px;
  font-size: 1rem;
  transition: all 0.3s ease;
  background: #f8f9fa;
  color: var(--text-color);
}

.form-group input:focus {
  border-color: var(--secondary-color);
  outline: none;
  box-shadow: 0 0 0 4px rgba(52, 152, 219, 0.1);
  background: white;
}

.form-group input::placeholder {
  color: var(--text-muted);
  opacity: 0.5;
}

.password-toggle {
  position: absolute;
  right: 1rem;
  top: calc(50% + 1rem);
  transform: translateY(-50%);
  background: none;
  border: none;
  cursor: pointer;
  padding: 0.5rem;
  color: var(--text-muted);
  transition: color 0.3s ease;
  display: flex;
  align-items: center;
  justify-content: center;
  z-index: 1;
}

.password-toggle:hover {
  color: var(--secondary-color);
}

.password-toggle svg {
  width: 20px;
  height: 20px;
}

.error-message {
  background-color: rgba(231, 76, 60, 0.1);
  color: var(--error-color);
  padding: 1rem;
  border-radius: 12px;
  margin-bottom: 1.5rem;
  font-size: 0.95rem;
  display: flex;
```

## APPENDIX

```
    align-items: center;
    gap: 0.8rem;
    border: 1px solid rgba(231, 76, 60, 0.2);
    animation: shake 0.5s ease-in-out;
  }

  @keyframes shake {
    0%, 100% { transform: translateX(0); }
    25% { transform: translateX(-5px); }
    75% { transform: translateX(5px); }
  }

  .error-message svg {
    color: var(--error-color);
    flex-shrink: 0;
  }

  .login-button {
    width: 100%;
    padding: 1rem;
    background: var(--gradient-primary);
    color: white;
    border: none;
    border-radius: 12px;
    font-size: 1rem;
    font-weight: 600;
    cursor: pointer;
    transition: all 0.3s ease;
    position: relative;
    overflow: hidden;
    margin-bottom: 1.5rem;
  }

  .login-button:hover {
    transform: translateY(-2px);
    box-shadow: 0 4px 12px rgba(52, 152, 219, 0.2);
  }

  .login-button:active {
    transform: translateY(1px);
  }

  .login-button::after {
    content: "";
    position: absolute;
    top: 0;
    left: 0;
    width: 100%;
    height: 100%;
    background: linear-gradient(45deg, transparent, rgba(255,255,255,0.2), transparent);
    transform: translateX(-100%);
    transition: transform 0.6s ease;
  }

  .login-button:hover::after {
    transform: translateX(100%);
  }

  .login-link {
```

## APPENDIX

```
    text-align: center;
    color: var(--text-muted);
    font-size: 0.95rem;
}

.login-link a {
    color: var(--secondary-color);
    text-decoration: none;
    font-weight: 500;
    transition: color 0.3s ease;
    position: relative;
}

.login-link a::after {
    content: "";
    position: absolute;
    width: 100%;
    height: 2px;
    bottom: -2px;
    left: 0;
    background-color: var(--secondary-color);
    transform: scaleX(0);
    transform-origin: right;
    transition: transform 0.3s ease;
}

.login-link a:hover {
    color: var(--primary-color);
}

.login-link a:hover::after {
    transform: scaleX(1);
    transform-origin: left;
}

.decoration {
    position: absolute;
    width: 300px;
    height: 300px;
    background: linear-gradient(45deg, rgba(52, 152, 219, 0.1), rgba(44, 62, 80, 0.1));
    border-radius: 50%;
    z-index: 0;
    animation: float 6s ease-in-out infinite;
}

@keyframes float {
    0%, 100% { transform: translateY(0); }
    50% { transform: translateY(-20px); }
}

.decoration-1 {
    top: -150px;
    right: -150px;
    animation-delay: 0s;
}

.decoration-2 {
    bottom: -150px;
    left: -150px;
}
```



```

    animation-delay: -3s;
  }

  .remember-me {
    display: flex;
    align-items: center;
    gap: 0.5rem;
    margin-bottom: 1.5rem;
    color: var(--text-muted);
    font-size: 0.9rem;
  }

  .remember-me input[type="checkbox"] {
    width: 18px;
    height: 18px;
    border-radius: 4px;
    border: 2px solid var(--border-color);
    cursor: pointer;
    transition: all 0.3s ease;
  }

  .remember-me input[type="checkbox"]:checked {
    background-color: var(--secondary-color);
    border-color: var(--secondary-color);
  }

  @media (max-width: 480px) {
    .login-container {
      margin: 1rem;
      padding: 1.5rem;
    }

    .login-header h1 {
      font-size: 1.5rem;
    }

    .form-group input {
      font-size: 0.95rem;
      padding: 0.8rem;
    }
  }
</style>
</head>
<body>
  <div class="background-pattern"></div>
  <div class="login-container">
    <div class="decoration decoration-1"></div>
    <div class="decoration decoration-2"></div>

    <div class="login-header">
      <h1>
        <svg style="width:28px;height:28px" viewBox="0 0 24 24">
          <path fill="currentColor" d="M12,2A2,2 0 0,1 14,4C14,4.74 13.6,5.39
13,5.73V7H14A7,7 0 0,1 21,14H22A1,1 0 0,1 23,15V18A1,1 0 0,1 22,19H21V20A2,2 0 0,1
19,22H5A2,2 0 0,1 3,20V19H2A1,1 0 0,1 1,18V15A1,1 0 0,1 2,14H3A7,7 0 0,1
10,7H11V5.73C10.4,5.39 10,4.74 10,4A2,2 0 0,1 12,2M7.5,13A2.5,2.5 0 0,0 5,15.5A2.5,2.5 0 0,0
7.5,18A2.5,2.5 0 0,0 10,15.5A2.5,2.5 0 0,0 7.5,13M16.5,13A2.5,2.5 0 0,0 14,15.5A2.5,2.5 0 0,0
16.5,18A2.5,2.5 0 0,0 19,15.5A2.5,2.5 0 0,0 16.5,13Z" />
        </svg>
      </h1>
    </div>
  </div>

```

## APPENDIX

```

    Login
  </h1>
  <p>Welcome to Multi-Fuzzer AI Chatbot System</p>
</div>

<?php if ($error): ?>
  <div class="error-message">
    <svg style="width:20px;height:20px" viewBox="0 0 24 24">
      <path fill="currentColor" d="M13 14H11V9H13M13 18H11V16H13M1 21H23L12 2L1
21Z" />
    </svg>
    <?php echo $error; ?>
  </div>
<?php endif; ?>

<form method="POST" action="<?php echo htmlspecialchars($_SERVER["PHP_SELF"]); ?>"
id="loginForm">
  <div class="form-group">
    <label for="username">Username</label>
    <input type="text" id="username" name="username" required autocomplete="username"
placeholder="Enter your username">
  </div>

  <div class="form-group">
    <label for="password">Password</label>
    <input type="password" id="password" name="password" required autocomplete="current-
password" placeholder="Enter your password">
    <button type="button" class="password-toggle" id="togglePassword">
      <svg viewBox="0 0 24 24">
        <path fill="currentColor" d="M12,9A3,3 0 0,1 15,12A3,3 0 0,1 12,15A3,3 0 0,1
9,12A3,3 0 0,1 12,9M12,4.5C17,4.5 21,27,7.61 23,12C21,27,16.39 17,19.5 12,19.5C7,19.5
2.73,16.39 1,12C2.73,7.61 7,4.5 12,4.5M12,2A10,10 0 0,0 2,12A10,10 0 0,0 12,22A10,10 0 0,0
22,12A10,10 0 0,0 12,2Z" />
      </svg>
    </button>
  </div>

  <div class="remember-me">
    <input type="checkbox" id="remember" name="remember">
    <label for="remember">Remember me</label>
  </div>

  <button type="submit" class="login-button">Log In</button>
</form>

<div class="login-link">
  Don't have an account? <a href="create_user.php">Register here</a>
</div>
</div>

<script>
const togglePassword = document.querySelector('#togglePassword');
const password = document.querySelector('#password');
const eyeIcon = togglePassword.querySelector('svg');

togglePassword.addEventListener('click', function (e) {
  const type = password.getAttribute('type') === 'password' ? 'text' : 'password';
  password.setAttribute('type', type);

```

## APPENDIX

```
        if (type === 'password') {
            eyeIcon.innerHTML = '<path fill="currentColor" d="M12,9A3,3 0 0,1 15,12A3,3 0 0,1
12,15A3,3 0 0,1 9,12A3,3 0 0,1 12,9M12,4.5C17,4.5 21.27,7.61 23,12C21.27,16.39 17,19.5
12,19.5C7,19.5 2.73,16.39 1,12C2.73,7.61 7,4.5 12,4.5M12,2A10,10 0 0,0 2,12A10,10 0 0,0
12,22A10,10 0 0,0 22,12A10,10 0 0,0 12,2Z" />';
        } else {
            eyeIcon.innerHTML = '<path fill="currentColor" d="M12,4.5C7,4.5 2.73,7.61
1,12C2.73,16.39 7,19.5 12,19.5C17,19.5 21.27,16.39 23,12C21.27,7.61 17,4.5 12,4.5M12,17A5,5 0
0,1 7,12A5,5 0 0,1 12,7A5,5 0 0,1 17,12A5,5 0 0,1 12,17M12,9A3,3 0 0,0 9,12A3,3 0 0,0 12,15A3,3 0
0,0 15,12A3,3 0 0,0 12,9Z" />';
        }
    });

    // Form validation
    document.getElementById('loginForm').addEventListener('submit', function(e) {
        const username = document.getElementById('username').value.trim();
        const password = document.getElementById('password').value;

        if (username.length < 3) {
            e.preventDefault();
            alert('Username must be at least 3 characters long');
            return false;
        }

        if (password.length < 6) {
            e.preventDefault();
            alert('Password must be at least 6 characters long');
            return false;
        }
    });

    document.querySelectorAll('.form-group input').forEach(input => {
        input.addEventListener('focus', function() {
            this.parentElement.classList.add('focused');
        });

        input.addEventListener('blur', function() {
            this.parentElement.classList.remove('focused');
        });
    });
</script>
</body>
</html>
```

### logout.php

```
<?php
session_start();

// Destroy all session data
session_destroy();

// Redirect to login page
header("Location: login.php");
exit();
?>
```

## APPENDIX

### settings.php

```
<?php
session_start();

if (!isset($_SESSION['user_id'])) {
    header("Location: login.php");
    exit();
}

$db_host = 'localhost';
$db_user = 'root';
$db_pass = '';
$db_name = 'chatbot_db';

$conn = new mysqli($db_host, $db_user, $db_pass, $db_name);

if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}

$sql = "CREATE TABLE IF NOT EXISTS user_2fa (
    user_id INT PRIMARY KEY,
    secret VARCHAR(32),
    enabled BOOLEAN DEFAULT FALSE,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (user_id) REFERENCES users(id)
)";
$conn->query($sql);

$success_message = "";
$error_message = "";

require_once 'vendor/autoload.php';
use RobThree\Auth\TwoFactorAuth;

$ga = new TwoFactorAuth('Multi-Fuzzer AI Chatbot');
$secret = "";

$sql = "SELECT * FROM user_2fa WHERE user_id = ?";
$stmt = $conn->prepare($sql);
$stmt->bind_param("i", $_SESSION['user_id']);
$stmt->execute();
$result = $stmt->get_result();
$twofa = $result->fetch_assoc();

if (!$twofa) {
    $secret = $ga->createSecret();
    $sql = "INSERT INTO user_2fa (user_id, secret) VALUES (?, ?)";
    $stmt = $conn->prepare($sql);
    $stmt->bind_param("is", $_SESSION['user_id'], $secret);
    $stmt->execute();
} else {
    $secret = $twofa['secret'];
}

// Handle 2FA verification
if (isset($_POST['verify_2fa'])) {
    $code = $_POST['verification_code'];
```

## APPENDIX

```
if ($ga->verifyCode($secret, $code, 2)) {
    $sql = "UPDATE user_2fa SET enabled = TRUE WHERE user_id = ?";
    $stmt = $conn->prepare($sql);
    $stmt->bind_param("i", $_SESSION['user_id']);
    $stmt->execute();
    $success_message = "Two-factor authentication enabled successfully!";
} else {
    $error_message = "Invalid verification code. Please try again.";
}
}

// Handle form submission
if ($_SERVER["REQUEST_METHOD"] == "POST") {
    $user_id = $_SESSION['user_id'];

    if (isset($_POST['current_password']) && isset($_POST['new_password'])) {
        $current_password = $_POST['current_password'];
        $new_password = $_POST['new_password'];
        $confirm_password = $_POST['confirm_password'];

        $sql = "SELECT password FROM users WHERE id = ?";
        $stmt = $conn->prepare($sql);
        $stmt->bind_param("i", $user_id);
        $stmt->execute();
        $result = $stmt->get_result();
        $user = $result->fetch_assoc();

        if (password_verify($current_password, $user['password'])) {
            if ($new_password === $confirm_password) {
                // Update password
                $hashed_password = password_hash($new_password, PASSWORD_DEFAULT);
                $update_sql = "UPDATE users SET password = ? WHERE id = ?";
                $update_stmt = $conn->prepare($update_sql);
                $update_stmt->bind_param("si", $hashed_password, $user_id);

                if ($update_stmt->execute()) {
                    $success_message = "Password updated successfully!";
                } else {
                    $error_message = "Error updating password.";
                }
            } else {
                $error_message = "New passwords do not match.";
            }
        } else {
            $error_message = "Current password is incorrect.";
        }
    }
}
?>

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Account Settings - Multi-Fuzzer AI Chatbot System</title>
    <style>
        :root {
            --primary-color: #2c3e50;
```

## APPENDIX

```
--secondary-color: #3498db;
--accent-color: #e74c3c;
--success-color: #2ecc71;
--warning-color: #f1c40f;
--background-color: #f4f6f9;
--border-color: #e0e0e0;
--text-color: #2c3e50;
--text-muted: #7f8c8d;
--shadow-sm: 0 2px 4px rgba(0,0,0,0.05);
--shadow-md: 0 4px 6px rgba(0,0,0,0.1);
--shadow-lg: 0 10px 15px rgba(0,0,0,0.1);
}

body {
  font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
  background-color: var(--background-color);
  margin: 0;
  padding: 20px;
  min-height: 100vh;
  color: var(--text-color);
  line-height: 1.6;
}

.settings-container {
  max-width: 1000px;
  margin: 0 auto;
  background: white;
  border-radius: 12px;
  box-shadow: var(--shadow-lg);
  overflow: hidden;
}

.settings-header {
  background: linear-gradient(135deg, var(--primary-color), var(--secondary-color));
  color: white;
  padding: 2rem;
  position: relative;
}

.settings-header h1 {
  margin: 0;
  font-size: 1.8rem;
  font-weight: 600;
}

.settings-header p {
  margin: 5px 0 0;
  opacity: 0.9;
  font-size: 0.95rem;
}

.back-button {
  position: absolute;
  right: 2rem;
  top: 50%;
  transform: translateY(-50%);
  text-decoration: none;
  color: white;
  display: flex;
```

## APPENDIX

```
    align-items: center;
    gap: 8px;
    padding: 8px 16px;
    border-radius: 6px;
    background: rgba(255, 255, 255, 0.2);
    transition: all 0.3s ease;
}

.back-button:hover {
    background: rgba(255, 255, 255, 0.3);
    transform: translateY(-50%) translateX(-2px);
}

.settings-nav {
    background: #f8f9fa;
    padding: 1rem 2rem;
    border-bottom: 1px solid var(--border-color);
}

.nav-tabs {
    display: flex;
    gap: 2rem;
    margin: 0;
    padding: 0;
    list-style: none;
}

.nav-tabs li a {
    text-decoration: none;
    color: var(--text-muted);
    padding: 0.5rem 0;
    display: block;
    border-bottom: 2px solid transparent;
    transition: all 0.3s ease;
}

.nav-tabs li a.active {
    color: var(--secondary-color);
    border-bottom-color: var(--secondary-color);
}

.nav-tabs li a:hover {
    color: var(--secondary-color);
}

.settings-content {
    padding: 2rem;
}

.section {
    background: white;
    border-radius: 8px;
    padding: 1.5rem;
    margin-bottom: 2rem;
    border: 1px solid var(--border-color);
}

.section-header {
    display: flex;
```

## APPENDIX

```
    justify-content: space-between;
    align-items: center;
    margin-bottom: 1.5rem;
}

.section-title {
    font-size: 1.2rem;
    font-weight: 600;
    color: var(--primary-color);
    margin: 0;
}

.form-group {
    margin-bottom: 1.5rem;
}

.form-group label {
    display: block;
    margin-bottom: 0.5rem;
    color: var(--text-color);
    font-weight: 500;
}

.form-group input {
    width: 100%;
    padding: 0.8rem;
    border: 2px solid var(--border-color);
    border-radius: 6px;
    font-size: 0.95rem;
    transition: all 0.3s ease;
    box-sizing: border-box;
}

.form-group input:focus {
    border-color: var(--secondary-color);
    outline: none;
    box-shadow: 0 0 3px rgba(52, 152, 219, 0.1);
}

.password-strength {
    margin-top: 0.5rem;
    font-size: 0.85rem;
    color: var(--text-muted);
}

.strength-meter {
    height: 4px;
    background: #eee;
    margin-top: 0.5rem;
    border-radius: 2px;
    overflow: hidden;
}

.strength-meter div {
    height: 100%;
    width: 0;
    transition: all 0.3s ease;
}
```



## APPENDIX

```
.button-group {
  display: flex;
  gap: 1rem;
  margin-top: 2rem;
}

.button {
  padding: 0.8rem 1.5rem;
  border-radius: 6px;
  font-size: 0.95rem;
  font-weight: 600;
  cursor: pointer;
  transition: all 0.3s ease;
  border: none;
}

.button-primary {
  background-color: var(--secondary-color);
  color: white;
}

.button-primary:hover {
  background-color: #2980b9;
  transform: translateY(-2px);
}

.button-secondary {
  background-color: #f8f9fa;
  color: var(--text-color);
  border: 1px solid var(--border-color);
}

.button-secondary:hover {
  background-color: #e9ecef;
}

.message {
  padding: 1rem;
  border-radius: 6px;
  margin-bottom: 1.5rem;
  display: flex;
  align-items: center;
  gap: 0.5rem;
}

.success {
  background-color: #d4edda;
  color: #155724;
  border: 1px solid #c3e6cb;
}

.error {
  background-color: #f8d7da;
  color: #721c24;
  border: 1px solid #f5c6cb;
}

.info-text {
  font-size: 0.9rem;
}
```

## APPENDIX

```
    color: var(--text-muted);
    margin-top: 0.5rem;
}

@media (max-width: 768px) {
    .settings-container {
        margin: 0;
        border-radius: 0;
    }

    .settings-header {
        padding: 1.5rem;
    }

    .back-button {
        top: 1rem;
        right: 1rem;
        transform: none;
    }

    .nav-tabs {
        gap: 1rem;
        overflow-x: auto;
        padding-bottom: 0.5rem;
    }

    .settings-content {
        padding: 1rem;
    }
}

.requirements-list {
    list-style: none;
    padding: 0;
    margin: 10px 0;
    font-size: 0.85rem;
    color: var(--text-muted);
}

.requirement-item {
    display: flex;
    align-items: center;
    margin: 5px 0;
}

.requirement-item::before {
    content: "•";
    color: var(--accent-color);
    margin-right: 8px;
}

.password-strength {
    margin-top: 0.5rem;
}

.strength-meter {
    height: 4px;
    background: #eee;
    margin-top: 0.5rem;
}
```

## APPENDIX

```
    border-radius: 2px;
    overflow: hidden;
}

.strength-meter div {
    height: 100%;
    width: 0;
    transition: all 0.3s ease;
}

#strength-text {
    display: block;
    margin-top: 0.5rem;
    font-size: 0.85rem;
    color: var(--text-muted);
}

.setup-steps {
    margin: 1.5rem 0;
    padding-left: 1.5rem;
}

.setup-steps li {
    margin-bottom: 1rem;
}

.qr-code {
    margin: 1rem 0;
    padding: 1rem;
    background: white;
    border: 1px solid var(--border-color);
    border-radius: 6px;
    display: inline-block;
}

.secret-key {
    margin: 1rem 0;
    padding: 0.5rem;
    background: #f8f9fa;
    border-radius: 4px;
    font-family: monospace;
}

.verification-form {
    margin-top: 2rem;
}

.status-badge {
    display: inline-flex;
    align-items: center;
    gap: 0.5rem;
    padding: 0.5rem 1rem;
    border-radius: 20px;
    font-weight: 500;
    margin-bottom: 1rem;
}

.status-badge.enabled {
    background-color: #d4edda;
```

## APPENDIX

```
        color: #155724;
    }

    .status-badge svg {
        color: #2ecc71;
    }
</style>
</head>
<body>
    <div class="settings-container">
        <div class="settings-header">
            <h1>Account Settings</h1>
            <p>Manage your account preferences and security settings</p>
            <a href="index.php" class="back-button">
                <svg style="width:20px;height:20px" viewBox="0 0 24 24">
                    <path fill="currentColor"
d="M20,11V13H8L13.5,18.5L12.08,19.92L4.16,12L12.08,4.08L13.5,5.5L8,11H20Z" />
                </svg>
                Back to Dashboard
            </a>
        </div>

        <div class="settings-nav">
            <ul class="nav-tabs">
                <li><a href="#security" class="active">Security</a></li>
            </ul>
        </div>

        <div class="settings-content">
            <?php if ($success_message): ?>
                <div class="message success">
                    <svg style="width:24px;height:24px" viewBox="0 0 24 24">
                        <path fill="currentColor" d="M12 2C6.5 2 2 6.5 2 12S6.5 22 12 22 17.5 22 17.5 2
12 2M10 17L5 12L6.41 10.59L10 14.17L17.59 6.58L19 8L10 17Z" />
                    </svg>
                    <?php echo $success_message; ?>
                </div>
            <?php endif; ?>

            <?php if ($error_message): ?>
                <div class="message error">
                    <svg style="width:24px;height:24px" viewBox="0 0 24 24">
                        <path fill="currentColor" d="M13 14H11V9H13M13 18H11V16H13M1 21H23L12
2L1 21Z" />
                    </svg>
                    <?php echo $error_message; ?>
                </div>
            <?php endif; ?>

            <div class="section">
                <div class="section-header">
                    <h2 class="section-title">Change Password</h2>
                </div>
                <form method="POST" id="passwordForm">
                    <div class="form-group">
                        <label for="current_password">Current Password</label>
                        <input type="password" id="current_password" name="current_password" required>
                    </div>
                    <div class="form-group">
```

```

<label for="new_password">New Password</label>
<input type="password" id="new_password" name="new_password" required>
<div class="password-strength">
  <div class="strength-meter">
    <div id="strength-bar"></div>
  </div>
  <span id="strength-text">Password strength: Too weak</span>
  <ul id="password-requirements" class="requirements-list"></ul>
</div>
</div>
<div class="form-group">
  <label for="confirm_password">Confirm New Password</label>
  <input type="password" id="confirm_password" name="confirm_password" required>
  <p class="info-text">Password must be at least 8 characters long and include numbers,
letters, and special characters.</p>
</div>
<div class="button-group">
  <button type="submit" class="button button-primary">Update Password</button>
  <button type="reset" class="button button-secondary">Cancel</button>
</div>
</form>
</div>

<div class="section">
  <div class="section-header">
    <h2 class="section-title">Two-Factor Authentication</h2>
  </div>
  <?php if (!$twofa || !$twofa['enabled']): ?>
    <div class="2fa-setup">
      <p>Follow these steps to enable two-factor authentication:</p>
      <ol class="setup-steps">
        <li>Download and install Google Authenticator on your mobile device</li>
        <li>Scan this QR code with Google Authenticator:
          <div class="qr-code">
            
          </div>
          <p class="secret-key">Secret key: <code><?php echo $secret; ?></code></p>
        </li>
        <li>Enter the 6-digit code from Google Authenticator to verify</li>
      </ol>
      <form method="POST" class="verification-form">
        <div class="form-group">
          <label for="verification_code">Verification Code</label>
          <input type="text" id="verification_code" name="verification_code"
            pattern="[0-9]{6}" maxlength="6" required>
        </div>
        <button type="submit" name="verify_2fa" class="button button-primary">
          Verify and Enable 2FA
        </button>
      </form>
    </div>
  <?php else: ?>
    <div class="2fa-status">
      <div class="status-badge enabled">
        <svg style="width:20px;height:20px" viewBox="0 0 24 24">
          <path fill="currentColor" d="M12 2C6.5 2 2 6.5 2 12S6.5 22 12 22 17.5 22 12
17.5 2 12 2M10 17L5 12L6.41 10.59L10 14.17L17.59 6.58L19 8L10 17Z" />
        </svg>
      </div>
    </div>
  </div>

```

```

        Two-Factor Authentication Enabled
    </div>
    <p>Your account is protected with two-factor authentication.</p>
</div>
<?php endif; ?>
</div>
</div>
</div>
</div>

<script>
function checkPasswordStrength(password) {
    let strength = 0;
    let feedback = [];

    if (password.length >= 8) {
        strength += 20;
    } else {
        feedback.push("At least 8 characters");
    }

    if (password.match(/[0-9]+/)) {
        strength += 20;
    } else {
        feedback.push("At least one number");
    }

    if (password.match(/[a-z]+/)) {
        strength += 20;
    } else {
        feedback.push("At least one lowercase letter");
    }

    if (password.match(/[A-Z]+/)) {
        strength += 20;
    } else {
        feedback.push("At least one uppercase letter");
    }

    if (password.match(/[$@#&!%*?&]+/)) {
        strength += 20;
    } else {
        feedback.push("At least one special character ($@#&!%*?&)");
    }

    return {
        strength: strength,
        feedback: feedback
    };
}

document.getElementById('new_password').addEventListener('input', function(e) {
    const password = e.target.value;
    const result = checkPasswordStrength(password);
    const strengthBar = document.getElementById('strength-bar');
    const strengthText = document.getElementById('strength-text');

    strengthBar.style.width = result.strength + '%';

    strengthBar.style.backgroundColor =

```

```

    result.strength <= 20 ? '#e74c3c' :
    result.strength <= 40 ? '#f39c12' :
    result.strength <= 60 ? '#f1c40f' :
    result.strength <= 80 ? '#3498db' :
    '#2ecc71';

    let strengthLabel =
    result.strength <= 20 ? 'Very Weak' :
    result.strength <= 40 ? 'Weak' :
    result.strength <= 60 ? 'Medium' :
    result.strength <= 80 ? 'Strong' :
    'Very Strong';

    strengthText.innerHTML = `Password strength: <strong>${strengthLabel}</strong>`;

    const requirementsList = document.getElementById('password-requirements');
    if (result.feedback.length > 0) {
        requirementsList.innerHTML = result.feedback.map(item =>
            `<li class="requirement-item">${item}</li>`
        ).join("");
        requirementsList.style.display = 'block';
    } else {
        requirementsList.style.display = 'none';
    }
});

// Form validation
document.getElementById('passwordForm').addEventListener('submit', function(e) {
    const newPassword = document.getElementById('new_password').value;
    const confirmPassword = document.getElementById('confirm_password').value;
    const result = checkPasswordStrength(newPassword);

    if (newPassword !== confirmPassword) {
        e.preventDefault();
        alert('New passwords do not match!');
        return;
    }

    if (result.strength < 100) {
        e.preventDefault();
        alert('Password does not meet all requirements:\n' + result.feedback.join("\n"));
        return;
    }
});
</script>
</body>
</html>

```

## APPENDIX

### testing.php

```
<?php
session_start();

$valid_username = 'admin';
$valid_password = 'admin1234';

if (!isset($_SERVER['PHP_AUTH_USER']) ||
    $_SERVER['PHP_AUTH_USER'] !== $valid_username ||
    $_SERVER['PHP_AUTH_PW'] !== $valid_password) {
    header('WWW-Authenticate: Basic realm="Restricted Access"');
    header('HTTP/1.0 401 Unauthorized');
    echo 'Authentication required';
    exit;
}

error_reporting(E_ALL);
ini_set('display_errors', 1);

$output = "";
$error = "";
$auto_command = "";

if (isset($_FILES['security_file'])) {
    $filename = pathinfo($_FILES["security_file"]["name"], PATHINFO_FILENAME);
    $auto_command = "RUN_PARALLEL 30 " . $filename;
    error_log("Command generated for file: " . $filename);
}

if (isset($_GET['command'])) {
    $auto_command = htmlspecialchars($_GET['command']);
}

if ($_SERVER['REQUEST_METHOD'] === 'POST' && isset($_POST['command'])) {
    $command = trim($_POST['command']);

    $command = escapeshellcmd($command);

    $timestamp = date('Y-m-d_H-i-s');

    // Check for stop fuzzing command
    if (strcasecmp($command, 'STOP_FUZZING') === 0) {
        $diagnostic_command = "cd /home/einjun && /home/einjun/stop_fuzzing.sh";
    }
    // Check for parallel fuzzing command
    elseif (preg_match('/^RUN_PARALLEL\s+(\d+)\s+(\w+)\$/i', $command, $matches)) {
        $minutes = (int)$matches[1];
        $target = $matches[2];
        $diagnostic_command = "rm -rf /home/einjun/Desktop/outputhonggfuzz/* && cd /home/einjun
&& /home/einjun/run_parallel.sh {$minutes} {$target}";
    }
    // Check for AFL++ command
    elseif (preg_match('/^RUN_AFL_TIMED\s+(\d+)\s+(\w+)\$/i', $command, $matches)) {
        $minutes = (int)$matches[1];
        $target = $matches[2];
        $diagnostic_command = "cd /home/einjun && /home/einjun/run_afl.sh {$minutes} {$target}";
    }
    // Check for Honggfuzz command
```



## APPENDIX

```
elseif (preg_match('/^RUN_HONGGFUZZ\s+(\d+)\s+(\w+)\$/i', $command, $matches)) {
    $minutes = (int)$matches[1];
    $target = $matches[2];
    $diagnostic_command = "rm -rf /home/einjun/Desktop/outhonggfuzz/* && cd /home/einjun
&& /home/einjun/run_honggfuzz.sh {$minutes} {$target}";
} else {
    $diagnostic_command = "cd /home/einjun && export
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/home/einjun/AFLplusplus && export
HOME=/home/einjun && " . $command;
}

$output_array = array();
$return_status = 0;

$debug_info = "Debug Info:\n";
$debug_info .= "Current PHP User: " . exec('whoami') . "\n";
$debug_info .= "Current PHP Path: " . exec('pwd') . "\n";
$debug_info .= "Executing command: " . $diagnostic_command . "\n";

$sudo_command = "/usr/bin/sudo -n -u einjun /bin/bash -c \"'\" . $diagnostic_command . "\" 2>&1";

exec($sudo_command, $output_array, $return_status);

if ($return_status === 0) {
    $output = $debug_info . "\nCommand Output:\n" . implode("\n", $output_array);
} else {
    $error = $debug_info . "\nCommand execution failed with status: " . $return_status . "\nOutput: " .
implode("\n", $output_array);
}
}
?>

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Ubuntu CLI Web Interface</title>
    <style>
        :root {
            --primary-color: #2c3e50;
            --secondary-color: #3498db;
            --accent-color: #e74c3c;
            --background-color: #1e1e1e;
            --chat-bg: #2d2d2d;
            --text-color: #ffffff;
            --border-radius: 12px;
        }

        body {
            font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
            margin: 0;
            padding: 20px;
            background: var(--background-color);
            color: var(--text-color);
            min-height: 100vh;
        }

        .container {
```

## APPENDIX

```
    max-width: 1200px;
    margin: 0 auto;
}

h1 {
    color: var(--text-color);
    margin-bottom: 1.5rem;
    font-size: 1.8rem;
    font-weight: 600;
}

.terminal {
    background: #000;
    padding: 20px;
    border-radius: var(--border-radius);
    margin: 20px 0;
    white-space: pre-wrap;
    font-family: monospace;
    box-shadow: 0 4px 6px rgba(0, 0, 0, 0.1);
}

input[type="text"] {
    width: 100%;
    padding: 12px;
    margin: 10px 0;
    background: var(--chat-bg);
    border: 1px solid #444;
    color: var(--text-color);
    border-radius: var(--border-radius);
    font-size: 1rem;
}

input[type="submit"] {
    padding: 12px 24px;
    background: var(--secondary-color);
    border: none;
    color: white;
    cursor: pointer;
    border-radius: var(--border-radius);
    font-weight: 600;
    transition: all 0.3s ease;
}

input[type="submit"]:hover {
    background: var(--primary-color);
    transform: translateY(-2px);
}

.error {
    color: var(--accent-color);
}

.help-text {
    background: var(--chat-bg);
    padding: 20px;
    border-radius: var(--border-radius);
    margin: 20px 0;
    box-shadow: 0 4px 6px rgba(0, 0, 0, 0.1);
}
```

```

.help-text h3 {
  color: var(--secondary-color);
  margin-top: 0;
}

code {
  display: inline-block;
  background: rgba(0, 0, 0, 0.2);
  padding: 4px 8px;
  border-radius: 4px;
  margin: 4px 0;
  font-family: monospace;
}

/* Custom scrollbar */
::-webkit-scrollbar {
  width: 8px;
}

::-webkit-scrollbar-track {
  background: var(--chat-bg);
  border-radius: 4px;
}

::-webkit-scrollbar-thumb {
  background: var(--secondary-color);
  border-radius: 4px;
}

::-webkit-scrollbar-thumb:hover {
  background: var(--primary-color);
}

.command-form {
  display: flex;
  gap: 10px;
  margin: 20px 0;
}

.command-form input[type="text"] {
  flex: 1;
  margin: 0;
}

@media (max-width: 768px) {
  .command-form {
    flex-direction: column;
  }

  .command-form input[type="submit"] {
    width: 100%;
  }
}

.loading-overlay {
  position: fixed;
  top: 0;
  left: 0;
}

```

## APPENDIX

```
width: 100%;
height: 100%;
background: rgba(0, 0, 0, 0.85);
display: flex;
justify-content: center;
align-items: center;
z-index: 9999999;
}

.loading-status {
background: #2d2d2d;
color: #00ff00;
padding: 30px 50px;
border-radius: 10px;
text-align: center;
box-shadow: 0 0 20px rgba(0, 255, 0, 0.2);
max-width: 80%;
z-index: 999999;
}

.loading-spinner {
display: inline-block;
width: 50px;
height: 50px;
border: 4px solid #00ff00;
border-radius: 50%;
border-top-color: transparent;
animation: spin 1s linear infinite;
margin-bottom: 20px;
z-index: 999999;
}

@keyframes spin {
to {transform: rotate(360deg);}
}

.status-text {
font-size: 1.2em;
margin-bottom: 10px;
z-index: 999999;
}

.time-remaining {
color: #888;
font-size: 1em;
margin-top: 10px;
z-index: 999999;
}

#debugInfo {
position: fixed;
bottom: 20px;
left: 20px;
right: 20px;
max-height: 200px;
overflow-y: auto;
background: rgba(0, 0, 0, 0.8);
```

## APPENDIX

```
        z-index: 1001;
    }

    .back-button {
        position: absolute;
        top: 20px;
        right: 20px;
        background: #000000;
        color: #ffffff;
        border: none;
        padding: 12px 24px;
        border-radius: 6px;
        cursor: pointer;
        font-weight: 500;
        font-size: 14px;
        letter-spacing: 0.5px;
        text-decoration: none;
        display: inline-flex;
        align-items: center;
        gap: 8px;
        transition: all 0.3s ease;
        box-shadow: 0 2px 6px rgba(0, 0, 0, 0.15);
    }


    .back-button:before {
        content: "←";
        font-size: 16px;
    }


    .back-button:hover {
        background: #333333;
        transform: translateY(-1px);
        box-shadow: 0 4px 12px rgba(0, 0, 0, 0.2);
    }

    .back-button:active {
        background: #000000;
        transform: translateY(1px);
        box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1);
    }

    @media (max-width: 768px) {
        .back-button {
            padding: 10px 20px;
            font-size: 13px;
        }
    }
</style>

</head>
<body>
    <div class="container">
        <a href="index.php" class="back-button">Back</a>

        <h1>  Multi-Fuzzer Testing CLI Web Interface</h1>

        <div class="help-text">
            <h3>  Usage Examples:</h3>
            <p>Run AFL++ for specific duration and target:</p>
        </div>
    </div>
</body>
</html>
```

## APPENDIX

```
<code>RUN_AFL_TIMED 30 testsleep</code><br>
<code>RUN_AFL_TIMED 30 testkeyword</code><br>
<p>Run Honggfuzz only:</p>
<code>RUN_HONGGFUZZ 30 testsleep</code><br>
<code>RUN_HONGGFUZZ 30 testkeyword</code>
<p>Run AFL++ and Honggfuzz in parallel:</p>
<code>RUN_PARALLEL 30 testsleep</code><br>
<code>RUN_PARALLEL 30 testkeyword</code>
<p>Stop all fuzzing processes:</p>
<code>STOP_FUZZING</code>
<p>Regular commands:</p>
<code>ls</code><br>
<code>pwd</code>
</div>

<form method="POST" class="command-form">
  <input type="text" name="command" placeholder="Enter your command" value="<?php echo
htmlspecialchars($auto_command); ?>" required>
  <input type="submit" value="Execute">
</form>

<?php if ($output): ?>
  <div class="terminal">
    <?php echo htmlspecialchars($output); ?>
  </div>
<?php endif; ?>
</div>
</body>
</html>
```

**verify\_2fa.php**

```

<?php
session_start();
require_once 'vendor/autoload.php';
use RobThree\Auth\TwoFactorAuth;

if (!isset($_SESSION['requires_2fa'])) {
    header("Location: login.php");
    exit();
}

$error = "";
$tfa = new TwoFactorAuth('Multi-Fuzzer AI Chatbot System');

if ($_SERVER["REQUEST_METHOD"] == "POST") {
    $code = $_POST['code'];

    if ($tfa->verifyCode($_SESSION['2fa_secret'], $code)) {
        $_SESSION['user_id'] = $_SESSION['temp_user_id'];
        $_SESSION['username'] = $_SESSION['temp_username'];
        unset($_SESSION['temp_user_id']);
        unset($_SESSION['temp_username']);
        unset($_SESSION['requires_2fa']);
        unset($_SESSION['2fa_secret']);

        $success = "Verification successful! Redirecting to dashboard<span class='loading-
dots'></span>";
        header("refresh:2;url=index.php");
    } else {
        $error = "Invalid verification code. Please try again.";
    }
}

?>

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>2FA Verification - Multi-Fuzzer AI Chatbot System</title>
    <style>
        :root {
            --primary-color: #2c3e50;
            --secondary-color: #3498db;
            --background-color: #f4f6f9;
            --error-color: #e74c3c;
            --success-color: #2ecc71;
            --text-color: #2c3e50;
            --text-muted: #7f8c8d;
            --border-color: #e0e0e0;
            --shadow-sm: 0 2px 4px rgba(0,0,0,0.05);
            --shadow-md: 0 4px 6px rgba(0,0,0,0.1);
            --shadow-lg: 0 8px 24px rgba(0,0,0,0.1);
            --gradient-primary: linear-gradient(135deg, var(--primary-color), var(--secondary-color));
        }

        * {
            margin: 0;

```

## APPENDIX

```
padding: 0;
box-sizing: border-box;
}

body {
font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
background-color: var(--background-color);
min-height: 100vh;
display: flex;
align-items: center;
justify-content: center;
padding: 20px;
color: var(--text-color);
line-height: 1.6;
}

.verify-container {
background: white;
padding: 2.5rem;
border-radius: 16px;
box-shadow: var(--shadow-lg);
width: 100%;
max-width: 420px;
position: relative;
overflow: hidden;
backdrop-filter: blur(10px);
border: 1px solid rgba(255, 255, 255, 0.2);
animation: slideUp 0.5s ease-out;
transition: transform 0.5s ease-out;
}

@keyframes slideUp {
from {
opacity: 0;
transform: translateY(20px);
}
to {
opacity: 1;
transform: translateY(0);
}
}

@keyframes slideOut {
from {
transform: translateX(0);
opacity: 1;
}
to {
transform: translateX(100%);
opacity: 0;
}
}

.verify-container.slide-out {
animation: slideOut 0.5s ease-out forwards;
}

.loading-spinner {
display: none;
}
```



## APPENDIX

```
    position: fixed;
    top: 50%;
    left: 50%;
    transform: translate(-50%, -50%);
    width: 80px;
    height: 80px;
    border: 6px solid var(--background-color);
    border-radius: 50%;
    border-top: 6px solid var(--secondary-color);
    border-right: 6px solid var(--primary-color);
    animation: spin 1s linear infinite;
    z-index: 99999;
    pointer-events: none;
}

@keyframes spin {
    0% { transform: translate(-50%, -50%) rotate(0deg); }
    100% { transform: translate(-50%, -50%) rotate(360deg); }
}

.loading-spinner.show {
    display: block;
}

.verify-header {
    text-align: center;
    margin-bottom: 2rem;
    position: relative;
}

.verify-header h2 {
    color: var(--text-color);
    font-size: 1.8rem;
    margin-bottom: 0.5rem;
    display: flex;
    align-items: center;
    justify-content: center;
    gap: 0.8rem;
    position: relative;
}

.verify-header svg {
    color: var(--secondary-color);
    filter: drop-shadow(0 2px 4px rgba(52, 152, 219, 0.2));
}

.form-group {
    margin-bottom: 1.8rem;
    position: relative;
}

.form-group label {
    display: block;
    margin-bottom: 0.8rem;
    color: var(--text-color);
    font-weight: 500;
    font-size: 0.95rem;
    transition: color 0.3s ease;
}
```

```

.form-group input {
  width: 100%;
  padding: 1rem;
  border: 2px solid var(--border-color);
  border-radius: 12px;
  font-size: 1.2rem;
  transition: all 0.3s ease;
  text-align: center;
  letter-spacing: 0.5em;
  background: #f8f9fa;
  color: var(--text-color);
  font-weight: 600;
}

.form-group input:focus {
  border-color: var(--secondary-color);
  outline: none;
  box-shadow: 0 0 4px rgba(52, 152, 219, 0.1);
  background: white;
}

.form-group input::placeholder {
  color: var(--text-muted);
  opacity: 0.5;
}

.error-message {
  background-color: rgba(231, 76, 60, 0.1);
  color: var(--error-color);
  padding: 1rem;
  border-radius: 12px;
  margin-bottom: 1.5rem;
  font-size: 0.95rem;
  display: flex;
  align-items: center;
  gap: 0.8rem;
  border: 1px solid rgba(231, 76, 60, 0.2);
  animation: shake 0.5s ease-in-out;
}

@keyframes shake {
  0%, 100% { transform: translateX(0); }
  25% { transform: translateX(-5px); }
  75% { transform: translateX(5px); }
}

.error-message svg {
  color: var(--error-color);
  flex-shrink: 0;
}

.verify-button {
  width: 100%;
  padding: 1rem;
  background: var(--gradient-primary);
  color: white;
  border: none;
  border-radius: 12px;
}

```

## APPENDIX

```
font-size: 1rem;
font-weight: 600;
cursor: pointer;
transition: all 0.3s ease;
position: relative;
overflow: hidden;
}

.verify-button:hover {
  transform: translateY(-2px);
  box-shadow: 0 4px 12px rgba(52, 152, 219, 0.2);
}

.verify-button:active {
  transform: translateY(1px);
}

.verify-button::after {
  content: "";
  position: absolute;
  top: 0;
  left: 0;
  width: 100%;
  height: 100%;
  background: linear-gradient(45deg, transparent, rgba(255,255,255,0.2), transparent);
  transform: translateX(-100%);
  transition: transform 0.6s ease;
}

.verify-button:hover::after {
  transform: translateX(100%);
}

.decoration {
  position: absolute;
  width: 300px;
  height: 300px;
  background: linear-gradient(45deg, rgba(52, 152, 219, 0.1), rgba(44, 62, 80, 0.1));
  border-radius: 50%;
  z-index: 0;
  animation: float 6s ease-in-out infinite;
}

@keyframes float {
  0%, 100% { transform: translateY(0); }
  50% { transform: translateY(-20px); }
}

.decoration-1 {
  top: -150px;
  right: -150px;
  animation-delay: 0s;
}

.decoration-2 {
  bottom: -150px;
  left: -150px;
  animation-delay: -3s;
}
```

```

.help-text {
  text-align: center;
  color: var(--text-muted);
  font-size: 0.9rem;
  margin-top: 1.5rem;
  padding: 1rem;
  background: rgba(52, 152, 219, 0.05);
  border-radius: 8px;
  border: 1px solid rgba(52, 152, 219, 0.1);
}

.timer {
  text-align: center;
  color: var(--text-muted);
  font-size: 0.85rem;
  margin-top: 0.5rem;
  font-weight: 500;
}

.success-message {
  background-color: rgba(46, 204, 113, 0.1);
  color: var(--success-color);
  padding: 1rem;
  border-radius: 12px;
  margin-bottom: 1.5rem;
  font-size: 0.95rem;
  display: flex;
  align-items: center;
  gap: 0.8rem;
  border: 1px solid rgba(46, 204, 113, 0.2);
  animation: fadeIn 0.5s ease-out;
}

@keyframes fadeIn {
  from { opacity: 0; transform: translateY(-10px); }
  to { opacity: 1; transform: translateY(0); }
}

@keyframes fadeOut {
  from { opacity: 1; }
  to { opacity: 0; }
}

.loading-dots::after {
  content: '.';
  animation: dots 1.5s steps(5, end) infinite;
}

@keyframes dots {
  0%, 20% { content: '.'; }
  40% { content: '..'; }
  60% { content: '...'; }
  80%, 100% { content: ''; }
}

.verify-container.fade-out {
  animation: fadeOut 1s ease-out forwards;
}

```

## APPENDIX

```
@media (max-width: 480px) {
    .verify-container {
        margin: 1rem;
        padding: 1.5rem;
    }

    .verify-header h2 {
        font-size: 1.5rem;
    }

    .form-group input {
        font-size: 1.1rem;
        padding: 0.8rem;
    }
}
</style>
</head>
<body>
    <div class="loading-spinner"></div>
    <div class="verify-container">
        <div class="decoration decoration-1"></div>
        <div class="decoration decoration-2"></div>

        <div class="verify-header">
            <h2>
                <svg style="width:28px;height:28px" viewBox="0 0 24 24">
                    <path fill="currentColor" d="M12,1L3,5V11C3,16.55 6.84,21.74 12,23C17.16,21.74
21,16.55 21,11V5L12,1M12,7C13.4,7 14.8,8.1 14.8,9.5V11C15.4,11 16,11.6 16,12.3V15.8C16,16.4
15.4,17 14.8,17H9.2C8.6,17 8,16.4 8,15.7V12.2C8,11.6 8.6,11 9.2,11V9.5C9.2,8.1 10.6,7
12,7M12,8.2C11.2,8.2 10.5,8.7 10.5,9.5V11H13.5V9.5C13.5,8.7 12.8,8.2 12,8.2Z" />
                </svg>
                Two-Factor Authentication
            </h2>
        </div>

        <?php if ($error): ?>
            <div class="error-message">
                <svg style="width:20px;height:20px" viewBox="0 0 24 24">
                    <path fill="currentColor" d="M13 14H11V9H13M13 18H11V16H13M1 21H23L12 2L1
21Z" />
                </svg>
                <?php echo $error; ?>
            </div>
        <?php endif; ?>

        <?php if (isset($success)): ?>
            <div class="success-message">
                <svg style="width:20px;height:20px" viewBox="0 0 24 24">
                    <path fill="currentColor" d="M12 2C6.5 2 2 6.5 2 12S6.5 22 12 22 17.5 22 17.5 2
12 2M10 17L5 12L6.41 10.59L10 14.17L17.59 6.58L19 8L10 17Z" />
                </svg>
                <?php echo $success; ?>
            </div>
        <?php endif; ?>

        <form method="POST" id="verifyForm">
            <div class="form-group">
                <label for="code">Enter the 6-digit code from your authenticator app:</label>
```

```

        <input type="text" id="code" name="code" required pattern="[0-9]{6}" maxlength="6"
placeholder="000000" autocomplete="off">
        <div class="timer" id="timer">Code expires in: <span id="countdown">30</span>s</div>
    </div>
    <button type="submit" class="verify-button">Verify Code</button>
</form>

<p class="help-text">
    Open your authenticator app and enter the code shown there.
    The code changes every 30 seconds.
</p>
</div>

<script>
    function updateTimer() {
        const now = new Date();
        const seconds = now.getSeconds();
        const remainingSeconds = 30 - (seconds % 30);
        document.getElementById('countdown').textContent = remainingSeconds;

        if (remainingSeconds <= 5) {
            document.getElementById('countdown').style.color = 'var(--error-color)';
            document.getElementById('timer').style.color = 'var(--error-color)';
        } else {
            document.getElementById('countdown').style.color = 'var(--text-muted)';
            document.getElementById('timer').style.color = 'var(--text-muted)';
        }
    }

    updateTimer();
    setInterval(updateTimer, 1000);

    // Form validation
    document.getElementById('verifyForm').addEventListener('submit', function(e) {
        const code = document.getElementById('code').value;
        if (!/^\d{6}$/.test(code)) {
            e.preventDefault();
            alert('Please enter a valid 6-digit code');
            return false;
        }
    });

    <?php if (isset($success)): ?>
        const container = document.querySelector('.verify-container');
        const spinner = document.querySelector('.loading-spinner');
        spinner.classList.add('show');

        setTimeout(() => {
            container.classList.add('slide-out');
        }, 500);
    <?php endif; ?>
</script>
</body>
</html>

```

## APPENDIX

### Fuzzing Scripts

#### **run\_afl.sh**

```
#!/bin/bash

# Check if both parameters exist
if [ $# -ne 2 ]; then
    echo "Usage: $0 <duration_in_minutes> <target_program>"
    echo "Example: $0 30 testsleep"
    exit 1
fi

DURATION=$1
TARGET_PROGRAM=$2
TIMESTAMP=$(date +%Y-%m-%d_%H-%M-%S)
OUTPUT_DIR="/home/einjun/AFLplusplus/output1_${TIMESTAMP}"
TARGET="/home/einjun/AFLplusplus/${TARGET_PROGRAM}"
LOG_FILE="${OUTPUT_DIR}_report.txt"

# Set input directory based on target program
if [ "$TARGET_PROGRAM" = "testkeyword" ]; then
    INPUT_DIR="/home/einjun/AFLplusplus/inputkeyword"
elif [ "$TARGET_PROGRAM" = "testsleep" ]; then
    INPUT_DIR="/home/einjun/AFLplusplus/input1"
elif [ "$TARGET_PROGRAM" = "testbuffer" ]; then
    INPUT_DIR="/home/einjun/AFLplusplus/input1"
elif [ "$TARGET_PROGRAM" = "testdivision" ]; then
    INPUT_DIR="/home/einjun/AFLplusplus/inputdivision"
elif [ "$TARGET_PROGRAM" = "testsigill" ]; then
    INPUT_DIR="/home/einjun/AFLplusplus/inputsigill"
elif [ "$TARGET_PROGRAM" = "testalrm" ]; then
    INPUT_DIR="/home/einjun/AFLplusplus/inputalrm"
elif [ "$TARGET_PROGRAM" = "testabort" ]; then
    INPUT_DIR="/home/einjun/AFLplusplus/inputabort"
elif [ "$TARGET_PROGRAM" = "testdisk" ]; then
    INPUT_DIR="/home/einjun/AFLplusplus/inputdisk"
elif [ "$TARGET_PROGRAM" = "testfile" ]; then
    INPUT_DIR="/home/einjun/AFLplusplus/inputdisk"
else
    INPUT_DIR="/home/einjun/AFLplusplus/input"
fi

# Create log file and start logging
{
    echo "AFL++ Fuzzing Report"
    echo "======"
    echo "Start Time: $(date '+%Y-%m-%d %H:%M:%S')"
    echo "Duration: $DURATION minutes"
    echo "Target Program: $TARGET_PROGRAM"
    echo "Output Directory: $OUTPUT_DIR"
    echo "======"
    echo
} > "$LOG_FILE"

# Check if target program exists
if [ ! -f "$TARGET" ]; then
    echo "Error: Target program $TARGET does not exist!" >> "$LOG_FILE"
    exit 1
fi
```

## APPENDIX

```
# Start AFL++
/home/einjun/AFLplusplus/afl-fuzz -Q -i "$INPUT_DIR" -o "$OUTPUT_DIR" -t 5000+ --
"$TARGET" &

AFL_PID=$!
echo "AFL++ process started with PID: $AFL_PID" >> "$LOG_FILE"

check_crashes() {
    if [ -d "$OUTPUT_DIR/default/crashes" ] && [ "$(ls -A $OUTPUT_DIR/default/crashes)" ]; then
    {
        echo "$(date '+%Y-%m-%d %H:%M:%S') - Crashes found!"
        echo "=====
        echo "Crash directory contents:"
        ls -la "$OUTPUT_DIR/default/crashes"
        echo

        echo "Crash file details:"
        echo "=====
        for crash_file in "$OUTPUT_DIR/default/crashes"/*; do
            if [ -f "$crash_file" ]; then
                echo "File: $(basename "$crash_file")"
                echo "Size: $(ls -l "$crash_file" | awk '{print $5}') bytes"
                echo "Hexdump of crash input:"
                xxd "$crash_file"
                echo "-----"
            fi
        done

        if [ -f "$OUTPUT_DIR/default/fuzzer_stats" ]; then
            echo
            echo "Fuzzer statistics at crash time:"
            echo "=====
            cat "$OUTPUT_DIR/default/fuzzer_stats"
        fi

        echo
        echo "Crash detection completed at: $(date '+%Y-%m-%d %H:%M:%S')
        echo "=====
    } >> "$LOG_FILE"

    echo "Crashes found! Check report at: $LOG_FILE"
    kill -9 $AFL_PID
    exit 0
fi
}

TOTAL_SECONDS=$((DURATION * 60))
ELAPSED_SECONDS=0

while [ $ELAPSED_SECONDS -lt $TOTAL_SECONDS ]; do
    sleep 5
    ((ELAPSED_SECONDS+=5))
    check_crashes
    ELAPSED_MINUTES=$((ELAPSED_SECONDS / 60))
    ELAPSED_SECONDS_MOD=$((ELAPSED_SECONDS % 60))
    echo "$(date '+%Y-%m-%d %H:%M:%S') - Fuzzing for ${ELAPSED_MINUTES}m
    ${ELAPSED_SECONDS_MOD}s out of ${DURATION}m" >> "$LOG_FILE"
done
```



## APPENDIX

```
{
  echo
  echo "$(date '+%Y-%m-%d %H:%M:%S') - Time limit reached"
  echo "Total duration: $DURATION minutes"
  echo "No crashes found"
  echo "===== "
} >> "$LOG_FILE"

echo "Fuzzing completed! Check report at: $LOG_FILE"
kill -9 $AFL_PID
```

## APPENDIX

### run\_honggfuzz.sh

```
#!/bin/bash

# Check if both parameters exist
if [ $# -ne 2 ]; then
    echo "Usage: $0 <duration_in_minutes> <target_program>"
    echo "Example: $0 30 testsleep"
    exit 1
fi

DURATION=$1
TARGET_PROGRAM=$2
TIMESTAMP=$(date +%Y-%m-%d_%H-%M-%S)
HONGG FUZZ_OUTPUT_DIR="/home/einjun/Desktop/outputhonggfuzz/outputhonggfuzz_${TIMESTAMP}"
HONGG FUZZ_CRASH_DIR="/home/einjun/Desktop/outputhonggfuzz"

# Set input directory based on target program
if [ "$TARGET_PROGRAM" = "testkeyword" ]; then
    INPUT_DIR="/home/einjun/AFLplusplus/inputkeyword"
elif [ "$TARGET_PROGRAM" = "testsleep" ]; then
    INPUT_DIR="/home/einjun/AFLplusplus/input1"
elif [ "$TARGET_PROGRAM" = "testbuffer" ]; then
    INPUT_DIR="/home/einjun/AFLplusplus/input1"
elif [ "$TARGET_PROGRAM" = "testdivision" ]; then
    INPUT_DIR="/home/einjun/AFLplusplus/inputdivision"
elif [ "$TARGET_PROGRAM" = "testsigill" ]; then
    INPUT_DIR="/home/einjun/AFLplusplus/inputsigill"
elif [ "$TARGET_PROGRAM" = "testalrm" ]; then
    INPUT_DIR="/home/einjun/AFLplusplus/inputalrm"
elif [ "$TARGET_PROGRAM" = "testabort" ]; then
    INPUT_DIR="/home/einjun/AFLplusplus/inputabort"
elif [ "$TARGET_PROGRAM" = "testdisk" ]; then
    INPUT_DIR="/home/einjun/AFLplusplus/inputdisk"
elif [ "$TARGET_PROGRAM" = "testfile" ]; then
    INPUT_DIR="/home/einjun/AFLplusplus/inputdisk"
else
    INPUT_DIR="/home/einjun/AFLplusplus/input"
fi

TARGET="/home/einjun/AFLplusplus/${TARGET_PROGRAM}"
LOG_DIR="/home/einjun/AFLplusplus/logs"
LOG_FILE="${LOG_DIR}/honggfuzz_${TIMESTAMP}_report.txt"

mkdir -p "$LOG_DIR"
chmod 777 "$LOG_DIR"
chmod 777 "$HONGG FUZZ_CRASH_DIR"

# Create log file and start logging
{
    echo "Honggfuzz Fuzzing Report"
    echo "===== "
    echo "Start Time: $(date +%Y-%m-%d %H:%M:%S)"
    echo "Duration: $DURATION minutes"
    echo "Target Program: $TARGET_PROGRAM"
    echo "Output Directory: $HONGG FUZZ_OUTPUT_DIR"
    echo "===== "
    echo
```

## APPENDIX

```
} > "$LOG_FILE"

# Check if target program exists
if [ ! -f "$TARGET" ]; then
    echo "Error: Target program $TARGET does not exist!" >> "$LOG_FILE"
    exit 1
fi

check_crashes() {
    if [ -d "$HONGGFUZZ_OUTPUT_DIR" ]; then
        local honggfuzz_crashes=(
            "$HONGGFUZZ_CRASH_DIR"/HONGGFUZZ.REPORT.*
            "$HONGGFUZZ_CRASH_DIR"/SIGABRT.*
            "$HONGGFUZZ_CRASH_DIR"/SIGSEGV.*
            "$HONGGFUZZ_CRASH_DIR"/*.fuzz
        )

        for pattern in "${honggfuzz_crashes[@]"; do
            if ls $pattern 1> /dev/null 2>&1; then
                {
                    echo "====="
                    echo "Honggfuzz Crash Report"
                    echo "====="
                    echo "Time of Detection: $(date '+%Y-%m-%d %H:%M:%S')"
                    echo

                    for crash_file in $pattern; do
                        if [ -f "$crash_file" ]; then
                            echo "File: $(basename "$crash_file")"
                            echo "Size: $(ls -l "$crash_file" | awk '{print $5}') bytes"
                            echo "SHA256: $(sha256sum "$crash_file" | cut -d' ' -f1)"

                            if [[ "$crash_file" == *"REPORT"* ]]; then
                                echo "Report Content:"
                                tail -n 30 "$crash_file"
                            else
                                echo "Hexdump:"
                                xxd "$crash_file"

                                if command -v gdb &> /dev/null; then
                                    echo "Stack Trace:"
                                    echo -e "run < $crash_file\nbt\nquit" | gdb -q "$TARGET" 2>&1
                                fi
                            fi
                            echo "-----"
                        fi
                    done

                    if [ -f "$HONGGFUZZ_OUTPUT_DIR/HONGGFUZZ.STATS.txt" ]; then
                        echo "Honggfuzz Statistics:"
                        echo "-----"
                        cat "$HONGGFUZZ_OUTPUT_DIR/HONGGFUZZ.STATS.txt"
                    fi
                } >> "$LOG_FILE"

                echo "Crash found at $(date)" > "/home/einjun/AFLplusplus/crash_found.flag"
                chmod 777 "/home/einjun/AFLplusplus/crash_found.flag"

                return 0
            fi
        done
    fi
}
```

## APPENDIX

```
        fi
    done
fi
return 1
}

# Start Honggfuzz
echo "Starting Honggfuzz at $(date '+%Y-%m-%d %H:%M:%S')" >> "$LOG_FILE"
cd /home/einjun/Desktop/outhonggfuzz
honggfuzz -i "$INPUT_DIR" -o "$HONGGFUZZ_OUTPUT_DIR" -t 5 -s -- "$TARGET" &
HONGGFUZZ_PID=$!
echo "Honggfuzz process started with PID: $HONGGFUZZ_PID" >> "$LOG_FILE"

TOTAL_SECONDS=$((DURATION * 60))
ELAPSED_SECONDS=0

while [ $ELAPSED_SECONDS -lt $TOTAL_SECONDS ]; do
    sleep 3
    ((ELAPSED_SECONDS+=3))

    if [ $ELAPSED_SECONDS -ge 5 ]; then
        if check_crashes; then
            echo "Crashes found! Check report at: $LOG_FILE"
            kill -9 $HONGGFUZZ_PID
            exit 0
        fi
    fi

    ELAPSED_MINUTES=$((ELAPSED_SECONDS / 60))
    ELAPSED_SECONDS_MOD=$((ELAPSED_SECONDS % 60))
    echo "$(date '+%Y-%m-%d %H:%M:%S') - Fuzzing for ${ELAPSED_MINUTES}m
    ${ELAPSED_SECONDS_MOD}s out of ${DURATION}m" >> "$LOG_FILE"
done

{
    echo
    echo "Fuzzing completed (time limit reached)"
    echo "====="
    echo "Total runtime: $DURATION minutes"
    echo "End Time: $(date '+%Y-%m-%d %H:%M:%S')"
    echo "No crashes found"

    if [ -f "$HONGGFUZZ_OUTPUT_DIR/HONGGFUZZ.STATS" ]; then
        echo
        echo "Final Honggfuzz Statistics:"
        cat "$HONGGFUZZ_OUTPUT_DIR/HONGGFUZZ.STATS"
    fi

    echo "====="
} >> "$LOG_FILE"

echo "Honggfuzz fuzzing completed! Check report at: $LOG_FILE"
kill -9 $HONGGFUZZ_PID
```

## APPENDIX

### run\_parallel.sh

```
#!/bin/bash

# Check if both parameters exist
if [ $# -ne 2 ]; then
    echo "Usage: $0 <duration_in_minutes> <target_program>"
    echo "Example: $0 30 testsleep"
    exit 1
fi

DURATION=$1
TARGET_PROGRAM=$2
TIMESTAMP=$(date +%Y-%m-%d_%H-%M-%S)
AFL_OUTPUT_DIR="/home/einjun/AFLplusplus/output1_${TIMESTAMP}"
HONGGFUZZ_OUTPUT_DIR="/home/einjun/Desktop/puthonggfuzz/outputhonggfuzz_${TIMESTAMP}"
HONGGFUZZ_CRASH_DIR="/home/einjun/Desktop/puthonggfuzz"

# Set input directory based on target program
if [ "$TARGET_PROGRAM" = "testkeyword" ]; then
    INPUT_DIR="/home/einjun/AFLplusplus/inputkeyword"
elif [ "$TARGET_PROGRAM" = "testsleep" ]; then
    INPUT_DIR="/home/einjun/AFLplusplus/input1"
elif [ "$TARGET_PROGRAM" = "testbuffer" ]; then
    INPUT_DIR="/home/einjun/AFLplusplus/input1"
elif [ "$TARGET_PROGRAM" = "testdivision" ]; then
    INPUT_DIR="/home/einjun/AFLplusplus/inputdivision"
elif [ "$TARGET_PROGRAM" = "testsigill" ]; then
    INPUT_DIR="/home/einjun/AFLplusplus/inputsigill"
elif [ "$TARGET_PROGRAM" = "testalrm" ]; then
    INPUT_DIR="/home/einjun/AFLplusplus/inputalrm"
elif [ "$TARGET_PROGRAM" = "testabort" ]; then
    INPUT_DIR="/home/einjun/AFLplusplus/inputabort"
elif [ "$TARGET_PROGRAM" = "testdisk" ]; then
    INPUT_DIR="/home/einjun/AFLplusplus/inputdisk"
elif [ "$TARGET_PROGRAM" = "testfile" ]; then
    INPUT_DIR="/home/einjun/AFLplusplus/inputdisk"
else
    INPUT_DIR="/home/einjun/AFLplusplus/input"
fi

TARGET="/home/einjun/AFLplusplus/${TARGET_PROGRAM}"
LOG_DIR="/home/einjun/AFLplusplus/logs"
LOG_FILE="${LOG_DIR}/parallel_fuzzing_${TIMESTAMP}_report.txt"

mkdir -p "$LOG_DIR"
chmod 777 "$LOG_DIR"
chmod 777 "$HONGGFUZZ_CRASH_DIR"

# Create log file and start logging
{
    echo "Parallel Fuzzing Report (AFL++ and Honggfuzz)"
    echo "=====
    echo "Start Time: $(date +%Y-%m-%d %H:%M:%S)"
    echo "Duration: $DURATION minutes"
    echo "Target Program: $TARGET_PROGRAM"
    echo "AFL++ Output Directory: $AFL_OUTPUT_DIR"
    echo "Honggfuzz Output Directory: $HONGGFUZZ_OUTPUT_DIR"
```

## APPENDIX

```
    echo "=====
    echo
    echo "AFL++ Configuration:"
    echo "-----"
    echo "Input Directory: $INPUT_DIR"
    echo "Output Directory: $AFL_OUTPUT_DIR"
    echo "Command: afl-fuzz -Q -i $INPUT_DIR -o $AFL_OUTPUT_DIR -t 5000+ -- $TARGET"
    echo
    echo "Honggfuzz Configuration:"
    echo "-----"
    echo "Input Directory: $INPUT_DIR"
    echo "Output Directory: $HONGGFUZZ_OUTPUT_DIR"
    echo "Command: honggfuzz -i $INPUT_DIR -o $HONGGFUZZ_OUTPUT_DIR -t 5 -s --
$TARGET"
    echo "=====
    echo
} > "$LOG_FILE"

# Check if target program exists
if [ ! -f "$TARGET" ]; then
    echo "Error: Target program $TARGET does not exist!" >> "$LOG_FILE"
    exit 1
fi

# Get detailed stack trace
get_detailed_stack_trace() {
    local crash_file=$1
    local target=$2
    local output=""

    # Create GDB commands file
    cat > /tmp/gdb_commands <<EOF
set pagination off
set logging on
run < "${crash_file}"
bt full
info registers
x/16i $pc
print $_siginfo
info locals
info frame
info threads
EOF

    # Run GDB with commands and capture output
    output=$(gdb -q -batch -x /tmp/gdb_commands "$target" 2>&1)
    rm /tmp/gdb_commands
    echo "$output"
}

check_crashes() {
    local crash_detected=false
    local crash_source=""
    local crash_details=""

    # Check AFL++ crashes
    if [ -d "$AFL_OUTPUT_DIR/default/crashes" ] && [ "$(ls -A
$AFL_OUTPUT_DIR/default/crashes)" ]; then
        crash_detected=true
    fi
}
```

```

crash_source="AFL++"

crash_details+="=====\n"
crash_details+="AFL++ Crash Information\n"
crash_details+="=====\n"
crash_details+="Time of Detection: $(date '+%Y-%m-%d %H:%M:%S')\n\n"

# Add AFL++ crash files details
crash_details+="Crash Files Found:\n"
crash_details+="-----\n"
for crash_file in "$AFL_OUTPUT_DIR/default/crashes"/*; do
    if [ -f "$crash_file" ] && [ "$(basename "$crash_file")" != "README.txt" ]; then
        crash_details+="File: $(basename "$crash_file")\n"
        crash_details+="Size: $(ls -l "$crash_file" | awk '{print $5}') bytes\n"
        crash_details+="SHA256: $(sha256sum "$crash_file" | cut -d' ' -f1)\n"
        crash_details+="Hexdump:\n"
        crash_details+="$(xxd "$crash_file")\n"

        # Get stack trace
        if command -v gdb &> /dev/null; then
            crash_details+="Stack Trace:\n"
            crash_details+="$(echo -e "run < $crash_file\nbt\nquit" | gdb -q "$TARGET" 2>&1)\n"
        fi
        crash_details+="-----\n"
    fi
done

if [ -f "$AFL_OUTPUT_DIR/default/fuzzer_stats" ]; then
    crash_details+="\nAFL++ Fuzzer Statistics:\n"
    crash_details+="-----\n"
    crash_details+="$(cat "$AFL_OUTPUT_DIR/default/fuzzer_stats")\n"
fi
fi

# Check Honggfuzz crashes
if [ -d "$HONGGFUZZ_OUTPUT_DIR" ]; then
    local honggfuzz_crashes=(
        "$HONGGFUZZ_CRASH_DIR"/HONGGFUZZ.REPORT.*
        "$HONGGFUZZ_CRASH_DIR"/SIGABRT.*
        "$HONGGFUZZ_CRASH_DIR"/SIGSEGV.*
        "$HONGGFUZZ_CRASH_DIR"/*.fuzz
    )

    for pattern in "${honggfuzz_crashes[@]"; do
        if ls $pattern 1> /dev/null 2>&1; then
            crash_detected=true
            [ -z "$crash_source" ] && crash_source="Honggfuzz" || crash_source="Both Fuzzers"

            crash_details+="\n=====\n"
            crash_details+="Honggfuzz Crash Information\n"
            crash_details+="=====\n"
            crash_details+="Time of Detection: $(date '+%Y-%m-%d %H:%M:%S')\n\n"

            for crash_file in $pattern; do
                if [ -f "$crash_file" ]; then
                    crash_details+="File: $(basename "$crash_file")\n"
                    crash_details+="Size: $(ls -l "$crash_file" | awk '{print $5}') bytes\n"
                    crash_details+="SHA256: $(sha256sum "$crash_file" | cut -d' ' -f1)\n"
                fi
            done
        fi
    done
fi

```

## APPENDIX

```

if [[ "$crash_file" == *"REPORT"* ]]; then
    crash_details+="Report Content:\n"
    crash_details+="$(tail -n 30 "$crash_file")\n"
else
    crash_details+="Hexdump:\n"
    crash_details+="$(xxd "$crash_file")\n"
    crash_details+="Crash Reproduction:\n"
    crash_details+="$(($TARGET < "$crash_file" 2>&1))\n"

    # Get stack trace using GDB
    if command -v gdb &> /dev/null; then
        crash_details+="Stack Trace:\n"
        crash_details+="$(echo -e "run < $crash_file\nbt\nquit" | gdb -q "$TARGET"
2>&1)\n"
        fi
        fi
        crash_details+="-----\n"
        fi
    done

    # Add Honggfuzz statistics
    if [ -f "$HONGGFUZZ_OUTPUT_DIR/HONGGFUZZ.STATS.txt" ]; then
        crash_details+="\nHonggfuzz Statistics:\n"
        crash_details+="-----\n"
        crash_details+="$(cat "$HONGGFUZZ_OUTPUT_DIR/HONGGFUZZ.STATS.txt")\n"
        fi

    break
fi
done
fi

if [ "$crash_detected" = true ]; then
{
    echo "====="
    echo "Combined Fuzzing Crash Report"
    echo "====="
    echo "Crash Source: $crash_source"
    echo "Target Program: $TARGET_PROGRAM"
    echo "Total Runtime: ${ELAPSED_SECONDS} seconds"
    echo "Report Generated: $(date '+%Y-%m-%d %H:%M:%S')"
    echo
    echo -e "$crash_details"
    echo "====="
    echo "End of Report"
    echo "====="
    return 0
} >> "$LOG_FILE"

echo "Crash found at $(date)" > "/home/einjun/AFLplusplus/crash_found.flag"
chmod 777 "/home/einjun/AFLplusplus/crash_found.flag"

return 0
fi

return 1
}

# Start AFL++

```



## APPENDIX

```
echo "Starting AFL++ at $(date '+%Y-%m-%d %H:%M:%S')" >> "$LOG_FILE"
/home/einjun/AFLplusplus/afl-fuzz -Q -i "$INPUT_DIR" -o "$AFL_OUTPUT_DIR" -t 5000+ --
"$TARGET" &
AFL_PID=$!
echo "AFL++ process started with PID: $AFL_PID" >> "$LOG_FILE"

# Start Honggfuzz
echo "Starting Honggfuzz at $(date '+%Y-%m-%d %H:%M:%S')" >> "$LOG_FILE"
cd /home/einjun/Desktop/outhonggfuzz
honggfuzz -i "$INPUT_DIR" -o "$HONGGFUZZ_OUTPUT_DIR" -t 5 -s -- "$TARGET" &
HONGGFUZZ_PID=$!
echo "Honggfuzz process started with PID: $HONGGFUZZ_PID" >> "$LOG_FILE"

TOTAL_SECONDS=$((DURATION * 60))
ELAPSED_SECONDS=0

while [ $ELAPSED_SECONDS -lt $TOTAL_SECONDS ]; do
    sleep 3
    ((ELAPSED_SECONDS+=3))

    if [ $ELAPSED_SECONDS -ge 5 ]; then
        # Check for crashes
        if check_crashes; then
            echo "Crashes found! Check report at: $LOG_FILE" | tee -a "$LOG_FILE"
            {
                echo "Terminating fuzzers due to crash detection"
                echo "AFL++ PID: $AFL_PID"
                echo "Honggfuzz PID: $HONGGFUZZ_PID"
            } >> "$LOG_FILE"
            kill -9 $AFL_PID
            kill -9 $HONGGFUZZ_PID
            exit 0
        fi
    fi

    ELAPSED_MINUTES=$((ELAPSED_SECONDS / 60))
    ELAPSED_SECONDS_MOD=$((ELAPSED_SECONDS % 60))
    echo "$(date '+%Y-%m-%d %H:%M:%S') - Fuzzing for ${ELAPSED_MINUTES}m
    ${ELAPSED_SECONDS_MOD}s out of ${DURATION}m" >> "$LOG_FILE"
done

{
    echo
    echo "Fuzzing completed (time limit reached)"
    echo "====="
    echo "Total runtime: $DURATION minutes"
    echo "End Time: $(date '+%Y-%m-%d %H:%M:%S')"
    echo "No crashes found"

    if [ -f "$AFL_OUTPUT_DIR/default/fuzzer_stats" ]; then
        echo
        echo "Final AFL++ Statistics:"
        cat "$AFL_OUTPUT_DIR/default/fuzzer_stats"
    fi

    if [ -f "$HONGGFUZZ_OUTPUT_DIR/HONGGFUZZ.STATS" ]; then
        echo
        echo "Final Honggfuzz Statistics:"
        cat "$HONGGFUZZ_OUTPUT_DIR/HONGGFUZZ.STATS"
    fi
}
```

## APPENDIX

```
fi

echo "=====
"
} >> "$LOG_FILE"

echo "Parallel fuzzing completed! Check report at: $LOG_FILE"

kill -9 $AFL_PID
kill -9 $HONGGFUZZ_PID
```

## APPENDIX

### stop\_fuzzing.sh

```
#!/bin/bash
```

```
LOG_FILE="/home/einjun/stopscrip/fuzzing_stop_$(date +%Y-%m-%d_%H-%M-%S).log"
```

```
{
  echo "Stopping Fuzzing Processes"
  echo "=====
  echo "Time: $(date '+%Y-%m-%d %H:%M:%S')"
  echo

  kill_processes() {
    local process_name=$1
    local pids=$(pgrep -f "$process_name")

    if [ -n "$pids" ]; then
      echo "Found $process_name processes:"
      ps -fp $pids
      echo "Killing $process_name processes..."
      kill -9 $pids 2>/dev/null
      if [ $? -eq 0 ]; then
        echo "Successfully killed $process_name processes"
      else
        echo "Error killing some $process_name processes"
      fi
    else
      echo "No $process_name processes found"
    fi
    echo
  }
}
```

```
# Stop AFL++ processes
echo "Stopping AFL++ processes..."
kill_processes "afl-fuzz"
```

```
# Stop Honggfuzz processes
echo "Stopping Honggfuzz processes..."
kill_processes "honggfuzz"
```

```
echo "=====
echo "Stop operation completed"
echo "Time: $(date '+%Y-%m-%d %H:%M:%S')"
```

```
} | tee "$LOG_FILE"
```

```
echo "All fuzzing processes have been stopped. Check log at: $LOG_FILE"
```

## APPENDIX

### Mitigation Script

#### **mitigate\_disk\_flood\_attack.sh**

```
#!/bin/bash

set -e

echo "=====
echo "Starting Disk Flood Attack Mitigation Tool"
echo "=====

echo "Scanning for large files in /tmp..."
FILE="/tmp/fuzz_test"

echo "-----"
echo "Disk space before operation:"
df -h /tmp
echo "-----"

if [ -f "$FILE" ]; then
    echo "Setting permissions and ownership for safe deletion..."

    if ! sudo chmod 666 "$FILE" 2>/dev/null; then
        echo "Sudo failed, trying direct command..."
        chmod 666 "$FILE" || echo "Failed to set permissions"
    fi

    if ! sudo chown daemon:daemon "$FILE" 2>/dev/null; then
        echo "Sudo ownership change failed, continuing anyway..."
    fi

    echo "-----"
    echo "Removing $FILE to free up space..."
    if rm -f "$FILE" 2>/dev/null; then
        echo "Success: Disk space has been freed!"
    else
        echo "Warning: Failed to remove file. Trying alternative method..."
        echo "<?php unlink('$FILE'); echo 'PHP unlink result: ' . (file_exists('$FILE') ? 'Failed' :
'Success'); ?>" > /tmp/remove_file.php
        php /tmp/remove_file.php
        rm -f /tmp/remove_file.php
    fi
else
    echo "Status: No large files found."
fi

echo "-----"
echo "Disk space after operation:"
df -h /tmp
echo "-----"

echo "=====
echo "Disk Flood Attack Mitigation Complete"
echo "=====
```

## APPENDIX

### **mitigate\_inodes\_exhaustion.sh**

```
#!/bin/bash

TARGET_DIR="/tmp/inode_flood/"

echo "Checking inode usage before mitigation..."
stat -f /tmp

echo ""
echo "Removing excessive files to free inodes..."
rm -rf ${TARGET_DIR}

echo ""
echo "Checking inode usage after mitigation..."
stat -f /tmp

echo ""
echo "Mitigation complete."
```

### **install\_apparmor.sh**

```
#!/bin/bash

LOG_FILE="/var/log/security_install.log"

log_message() {
    echo "[$(date '+%Y-%m-%d %H:%M:%S')] $1" | tee -a "$LOG_FILE"
}

log_message "Starting AppArmor installation..."

export DEBIAN_FRONTEND=noninteractive

# Install AppArmor
log_message "Installing AppArmor packages..."
sudo apt-get update -y
sudo apt-get install -y apparmor apparmor-utils apparmor-profiles

log_message "Enabling AppArmor..."
sudo systemctl enable apparmor
sudo systemctl start apparmor

log_message "Checking AppArmor status..."
sudo mkdir -p /var/log/apparmor
sudo bash -c 'aa-status > /var/log/apparmor/status.txt'
sudo chmod 644 /var/log/apparmor/status.txt

log_message "AppArmor installation completed."
sudo mkdir -p /etc/apparmor
sudo touch /etc/apparmor/installed_flag

exit 0
```

## APPENDIX

### **install\_asan.sh**

```
#!/bin/bash

LOG_FILE="/var/log/security_install.log"

log_message() {
    echo "[$(date '+%Y-%m-%d %H:%M:%S')] $1" | tee -a "$LOG_FILE"
}

log_message "Starting AddressSanitizer installation..."

if [ "$EUID" -ne 0 ]; then
    log_message "Error: Please run as root"
    exit 1
fi

log_message "Installing LLVM and Clang..."
apt-get update
apt-get install -y clang llvm

log_message "Configuring ASan..."
cat > /etc/ld.so.conf.d/asan.conf << EOF
/usr/lib/llvm-11/lib/clang/11.0.0/lib/linux
EOF
ldconfig

log_message "AddressSanitizer installation completed."
touch /usr/lib/llvm/installed_flag

exit 0
```

### **install\_exploit\_mitigations.sh**

```
#!/bin/bash

LOG_FILE="/var/log/security_install.log"

log_message() {
    echo "[$(date '+%Y-%m-%d %H:%M:%S')] $1" | tee -a "$LOG_FILE"
}

log_message "Starting Exploit Mitigations installation..."

if [ "$EUID" -ne 0 ]; then
    log_message "Error: Please run as root"
    exit 1
fi

log_message "Enabling ASLR..."
echo 2 > /proc/sys/kernel/randomize_va_space
echo "kernel.randomize_va_space = 2" >> /etc/sysctl.conf

log_message "Configuring Stack Protection..."
echo "kernel.dmesg_restrict = 1" >> /etc/sysctl.conf
echo "kernel.kptr_restrict = 2" >> /etc/sysctl.conf
echo "kernel.yama.ptrace_scope = 2" >> /etc/sysctl.conf
```

## APPENDIX

```
log_message "Enabling NX bit..."
apt-get update
apt-get install -y execstack
find /usr/bin -type f -executable -exec execstack -c {} \;

log_message "Configuring additional security settings..."
cat >> /etc/sysctl.conf << EOF
kernel.exec-shield = 2
kernel.core_uses_pid = 1
kernel.sysrq = 0
net.ipv4.tcp_syncookies = 1
EOF

sysctl -p

log_message "Exploit Mitigations installation completed."
touch /etc/security/mitigations_installed_flag

exit 0
```

### **install\_hardened\_malloc.sh**

```
#!/bin/bash

LOG_FILE="/var/log/security_install.log"

log_message() {
    echo "[$(date '+%Y-%m-%d %H:%M:%S')] $1" | tee -a "$LOG_FILE"
}

log_message "Starting HardenedMalloc installation..."

if [ "$EUID" -ne 0 ]; then
    log_message "Error: Please run as root"
    exit 1
fi

log_message "Installing dependencies..."
apt-get update
apt-get install -y git build-essential

# Clone and build HardenedMalloc
log_message "Building HardenedMalloc..."
cd /usr/src
git clone https://github.com/GrapheneOS/hardened_malloc.git
cd hardened_malloc
make

log_message "Installing HardenedMalloc..."
cp out/libhardened_malloc.so /usr/lib/
echo "/usr/lib/libhardened_malloc.so" > /etc/ld.so.preload

log_message "HardenedMalloc installation completed."
touch /usr/lib/hardened_malloc/installed_flag

exit 0
```