# Process

## Setup and Execution

|  |  |
| --- | --- |
| sudo apt install afl**++** | *Install AFL++* |
| mkdir in out | *Create Input and Output Directories*  *in: input test cases*  *out: AFL++ output* |
| cp examples**/\*** in**/** | *Copy Example Test Cases* |
| make cc**=**afl**-**clang cflags**=**"-fsanitize=address -fsanitize=undefined -fsanitize=fuzzer-no-link" | *Compile with AFL++*  *address: Enables AddressSanitizer for memory error detection*  *undefined: Enables UndefinedBehavior Sanitizer for undefined behavior detection*  *fuzzer-no-link: Enables the fuzzer without link-time optimization* |
| sudo **-**i  echo core **>/**proc**/**sys**/**kernel**/**core\_pattern  logout | *Set Core Dump Pattern* |
| afl**-**fuzz **-**i in**/** **-**o out**/** **-**m none **--** **./**crashy**.**bin **@@** | *Run AFL++ Fuzzer*  *Note: command must include* -m none *to fix ‘*PROGRAM ABORT: Fork server crashed’ *error* [[1](https://github.com/ocaml-multicore/ocaml-multicore/issues/497)] |

## AFL++ Output

A computer screen shot of a program

Description automatically generated

# Writeup

For lab 10, I executed a fuzzing process to the "Crashy" program using AFL++. The goal was to identify crashes, with a particular focus on potential exploitable vulnerabilities, such as buffer overflows. As shown from the screenshot above, AFL++ found 7 unique crashes and 4 unique hangs. A list of the crashes can be seen below.

A screen shot of a computer program

Description automatically generated

However, after an analysis of these crashes, it was determined that they share a similar root cause. It is important to note that AFL++’s definition of ‘unique’ is more related to the exploration of the program's execution paths and diversity of inputs, rather than the specific nature of the crashes. As such, despite having similar causes, AFL++ treats them as unique since they take different code paths within the target program. Therefore, to maintain conciseness in this report, a breakdown of the vulnerability is consolidated and outlined below.

## Cause of Crashes

All identified crashes are rooted in a heap buffer overflow within the parse\_string function of the Crashy program. This vulnerability arises from the parse\_string function using strcpy (line 33) to copy the input string into a dynamically allocated buffer (str). However, the allocated buffer (str) is not large enough to hold the copied string, which leads to a heap buffer overflow.

## Crash Locations

Each crash occurs in the parse\_int function within the parse.c file, specifically at line 9 (src/parse.c:9:9).

A screen shot of a computer

Description automatically generated

## Crashing Sample

A computer screen shot of a computer program

Description automatically generated

Note: Crashing samples for each input are included in the folder for further analysis.