IDATT2503 - Exercise 6 - Edvard Berdal Eek

09.10.2025

Fix bugs you find through fuzzing, or introduce bugs that are discovered through fuzzing:

Initially i just replaced the method used in the LLVM fuzzer function. However, i quickly ran into issues as the fuzzer caused a **out-of-memory** error indicating a memory leak.

```
==26104== ERROR: libFuzzer: out-of-memory (used: 2129Mb; limit: 2048Mb)
   To change the out-of-memory limit use -rss_limit_mb=<N>

Live Heap Allocations: 1898670712 bytes in 10231825 chunks; quarantined: 48516926 bytes in 872992 chunks; 446588
 other chunks; total chunks: 11551405; showing top 95% (at most 8 unique contexts)
```

```c
int LLVMFuzzerTestOneInput(const uint8_t *data, size_t size) {
  // Create a c string from fuzzer data (with an additional byt
  char *str = (char *)malloc(sizeof(char) * size + 1); // Creat
  memcpy(str, data, size);                             // Copy
  str[size] = '\0';                                    // Set l

  replace_amp_lt_gt(str);

  free(str);

  return 0;
}
```

I realized that i had forgotten that the **replace_amp_lt_gt** function returns string with its memory allocated on the heap. Since i didnt free the memory allocated in the function it caused a memory leak.

I fixed this issue using **free** on the returned string pointer:

```c
int LLVMFuzzerTestOneInput(const uint8_t *data, size_t size) {
  // Create a c string from fuzzer data (with an additional byt
  char *str = (char *)malloc(sizeof(char) * size + 1); // Creat
  memcpy(str, data, size);                             // Copy
  str[size] = '\0';                                    // Set l

  char* fuzz = replace_amp_lt_gt(str);

  free(str);
  free(fuzz);

  return 0;
}
```

Now the fuzzer did not detect any errors. The function currently allocates the maximum amount of possible characters, incase all characters are ampersand which requires 4 more characters per occurance. So i multiplied the input length with 5 for the output length:

```
size_t len = strlen(input);
size_t max_len = len * 5 + 1;
char* output = malloc(max_len);
```

I then tried to introduce a bug where it did not accommodate this:

```
size_t len = strlen(input);
char* output = malloc(len);
```

This immediately led to a **heap-buffer-overflow**.

```
SUMMARY: AddressSanitizer: heap-buffer-overflow utility_fuzzer_test.c:12 in LLVMFuzzerTestOneInput
```

However, i know that the function does not validate that the input is not a null pointer. I did not find a way to make the fuzzer attempt NULL as input so i just added a check for it in the function:

```
if (input == NULL) return NULL;
```