Round 1:

Part 1:

- Time: 62 minutes
- Crash Inputs: 928 (17 unique)
- Hang Inputs: 0

Part2:

Crash ID2: “u 22222222222222222222#222H5”

- I started with this case because unlike ID0 and ID1 it instantly produced a segmentation fault.
- Using GDB, I found the segfault was caused by line 34, “out[i] = rest[i] - 32;”
- Thinking the counter must have gone beyond the bounds of the buffer rest, I changed the check on line 22 to use strlen(rest) instead of strlen(input). However, the segfault persisted.
- With some printf statements, I found the value of “len” was -1, indicating an overflow.
- I changed the type of “len” to “long” and removed the addition in the “out” buffer malloc call, replacing it with the length of “rest.” This fixes the bug because strtol() can return values that would overflow an int variable, but not a long variable. Removing the addition in the malloc call prevents overflow during the allocation.

Crash ID0: “u PPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPhead PP…”

- Now gives “*** Error in `./vulnerable': free(): invalid next size (fast): 0x0000000000809010 ***”
- Stepping through with GDB, I found the issue was at “free(out);” but caused because “len” was 0 (an error condition for strtol() triggered by the lack of numbers).
- I added a check for “len == 0” and moved the “out” buffer allocation after the length checking. This fixes the bug because the program will now return if strtol returns 0 instead of trying to allocate and mutate a buffer of length 0.

Crash ID15: “head 444444444444444 44…”

- Starts with a segmentation fault. GDB reveals it’s at “rest[len] = ‘\0’”
- This should be because “len” is larger than the length of “rest”
- I copied the length check from the “u” command section to ensure “len” is not longer than “rest.” This fixes the error because if the function returns before “rest[len]” is written to, no illegal memory access occurs.
- I also copied the check to make sure “len” is not 0, since I couldn’t think of a reason the validation for “len” should be different between “u” and “head.”

Crash ID1: "u 7░░"

- I have never seen this Unicode character before and I do not like it. Most of the remaining crashing inputs involve this character. I think the Unicode is encoded in too many bytes and is overflowing the input buffer. It also floods my terminal when it goes over the 100-byte limit.
- I rewrote the user input to use getline() to avoid overflowing into my terminal. I only copied INPUTSIZE-2 bytes to ensure there is always a newline and a null terminator.
    - I know this rewrite might not be necessary to prevent crashing, but it's a quality-of-life change for me, the inputs overflowing into the terminal were making the debugging process harder.
- This fixes the bug because I stopped the null terminator from being overwritten in the input buffer.

After those 4 fixes, all the unmentioned crash inputs no longer crashed.

Round 2:

Part 1:

- Time: 157 minutes
- Crash Inputs: 51 (5 unique)
- Hang Inputs: 0

Part 2:

- All the crashes for this round involved negative length inputs, for example "u -211 -211"
- The program segfaults at the line "out[i] = rest[i] - 32;" because the for-loop fails to terminate until after "i" overflows because "len" is negative, so out[i] or rest[i] try to access some very large offset.
- I added one more check for "len" in both "u" and "head" to ensure the user-inputted length is non-negative. This fixes the segfault because the program returns before the offending line.

Round 3:

Part 1:

- Time: 82 minutes
- Crash Inputs: 0
- Hang Inputs: 0

Part 2: No action was required as AFL did not reveal any new bugs.