Sam Dutson - u1406827

# CS 5493 Lab 2 : Crash Triage

## Overview(What I Fuzzed in Lab 1):

You can find the repo of what I fuzzed here:
https://github.com/mykter/afl-training/tree/main/quickstart

You can also find the results of my fuzzing experiments from Lab 1 here:
https://github.com/sdutson/CS5493_Lab1

## Setup:

For this lab I'll be using the crashes from when I ran afl++ with the following commands:

*CC=afl-clang-fast AFL_USE_ASAN=1 make clean all*
*afl-fuzz -i inputs -p explore -o out/combo_explore ./vulnerable*

The outputs from this run can be found in the 'combo_explore' subfolder in the repo that I linked above. I chose to use this run configuration because it found the most crashes out of any of the configurations I tried(22 total crashes in a 3.5 min run).

To create the ASAN-instrumented version of the target program I ran the following:

*CC=clang CFLAGS="-O1 -g -fsanitize=address -fno-omit-frame-pointer" make*

I then ran the code with the crash inputs from the '/combo_explore/default/crashes' folder(The inputs that AFL++ found would cause crashes). I copied the output from all these runs into the 'asan_reports' directory. This 'asan_reports' directory can be found at the 'CS5493_Lab1' repo linked above.

## Results:

I looked over all the ASAN output files and compiled the most 'interesting' information into the following table:

| Crash ID | Bug Type | Bug Location(Method : Line) | Bug Description | Input Bytes |
|---|---|---|---|---|
| 1 | heap-buffer-overflow | process : 39 | Tried to write 16 bytes into 8 byte region | 7520 3403 6300 0001 0a20 3403 5721 3230 |
| 2 | heap-buffer-overflow | process : 39 | Tried to write 5 bytes into 2 byte region | 7520 0001 e8 |
| 3 | heap-buffer-overflow | process : 36 | Tried to write 38 bytes into 37 byte region | 7520 3838 3838 3838 3838 3838 3838 3838 ... |
| 6 | heap-buffer-overflow | process : 36 | Tried to write 49 bytes into a 48 byte region | 7520 3232 3232 3232 3232 3232 3232 3232 ... |
| 8 | stack-buffer-overflow | process : 31 | Tried to read 81 bytes from a 80 byte buffer | 7520 3230 3030 3030 3030 3030 3030 3030 ... |
| 20 | stack-buffer-overflow | process : 39 | Tried to read 92 bytes from a 60 byte buffer | 7520 3634 2037 2063 6865 d6d6 d6d6 6561 ... |
| 21 | stack-buffer-overflow | process : 39 | Tried to read 81 bytes from a 80 byte buffer | 7520 3935 2037 2063 6865 d6d6 371a 636 |

**Note:** You'll notice that not all Crash IDs appear in the table above. I found that some crashes were very similar(They mapped to the same bugs). Therefore, I decided it would be redundant to add rows for those IDs. The above rows represent all the unique crashes that AFL++ found with the configuration specified in the 'Setup' section.

## Analysis:

1. **What sorts of bugs did your fuzzer find?**

   My fuzzer found heap-buffer-overflow and stack-buffer-overflow errors. All the crashes were associated with trying to read/write more bytes then there were in the associated region(See table above for more detail).

2. **Are you seeing lots of "unique" crashes map to only a few bugs? Or lots of bugs with a few crashes each?**

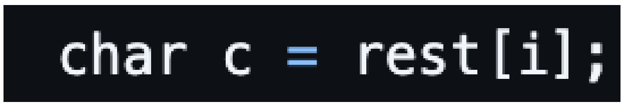
   I'm seeing lots of unique crashes mapping to only a few bugs. As shown in the table above, there appears to only be 3 unique bugs that were identified by the fuzzer(lines 31, 36, and 39 in the 'process' method). All 22 of the crashes can be directly mapped to one of those three bugs.

3. **What issues do you see, and what insights do you have?**
   Both heap-buffer-overflow and stack-buffer-overflow errors come with big security risks. As discussed in CS 4440, if an attacker can do unbounded writes they can redirect the execution of a program. The ‘vulnerable.c’ program that I fuzzed is SUPER susceptible to attack.

## What the Actual Bugs in the Code are:

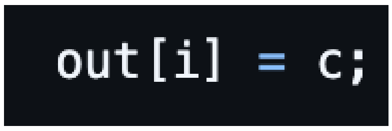
Below are the 3 bugs that the fuzzer found in the ‘vulnerable.c’ program.

vulnerable.c line 31(Stack overflow errors):



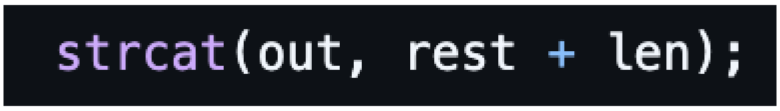
```
char c = rest[i];
```

vulnerable.c line 36(Heap overflow errors):



```
out[i] = c;
```

vulnerable.c line 39(Both stack and heap overflow errors):



```
strcat(out, rest + len);
```

**Sources:**
I used ChatGPT for terminal command syntax. I USED IT FOR SYNTAX ONLY! ALL WORK IN THIS PAPER IS 100% MINE.