White Box Testing

Palmer Cluff, Joshua Jolley, Austin Moore

# Defects

**Title:**: Seg Fault for String containing characters with an int value greater than 127

**Steps:**

1. Start the program
2. Hold alt
3. Enter the keycode for a character with an int value greater than 127 (e.g. entering 456 produces ╚)
4. Release alt
5. Hit enter

**Code:** Line 39. The character causes rToI[128] to step out of bounds.

**Title:** Seg Fault or std:bad\_alloc for values greater than MMMMMMMMMMM

**Steps:**

1. Start the program
2. Enter 11 or more Ms
3. Hit enter

**Code:** Line 63. 11000/1000 is greater than 10, which steps roman[4][10] out of bounds.

**Title:** Overwrite your stack by overstepping decimal[64] with strings > 66 in length

**Steps:**

1. Start the program
2. Enter a string that is 66 characters long.
3. Hit enter

**Code:** Line 39

**Note**: This code overwrites the variable i that you’re looping on. Depending on the final characters in your string you will either reset i to a value less than 66 and create an infinite loop, or push I past 66 and end the for loop, resulting in an invalid response.

**Title:** Incorrect response for values greater than 3999.

**Steps**:

1. Start the program
2. Enter the following: MMMM
3. Hit enter

**Code:** Line 58.

# Narratives

**Palmer:** Personally, I was able to find a bug by inspecting the code and brute-forcing my way to the solution. After I found that one bug was to input a string that was longer than 11 M’s it would seg-fault. As a group we met a little late in the week to try and get this assignment done. I think it would have been a little more beneficial to meet earlier in the week so that we could collaborate more and bounce ideas off of each other. Next week we will definitely start earlier.

**Jolley:** I started by attempting to write an sh script that would exercise the program for me, but ran into trouble when trying to pipe input into the program. It would create an infinite loop, which was bothersome. After a short while attempting to debug what was going on, I decided that my efforts were better spent elsewhere. I then wrote a debug macro (#define debug(x) cout << #x << ":" << x << "\n”) and used it in RomenNumerals.cpp to output the value of various variables during execution. I then noted the bounds of each array and attempted to enter malicious strings that would cause the program to overstep its bounds. With my debug information and knowledge of how the functions worked, I was able to overstep the bounds of each array. It took me a while to pin down what was happening when we overstepped decimal[64]. Our team had brief contact on Wed/Thu/Fri, but really dug in on Saturday. We had frequent but sporadic communication throughout Saturday. We communicated primarily over Google Doc’s chat, email, and Slack. If I were to do this again, I would take the additional time necessary to write good unit tests and dig into all of the corner cases. I also would start on the assignment earlier in the week, and would take the time to coordinate schedules.

**Austin:** I first went through the code function-by-function, trying to understand at a high level how the program worked. As the ‘Reader,’ I first focused on trying to find defects in the code that handled input. From there on, I moved on to the other functions. I found that trying to comment the code really helped. After gaining an understanding of the code, I stepped through it with a debugger. With the debugger, I was able to find that numbers 4000 (MMMM) and higher were not correctly classified as valid. Finally, with some fancy xargs usage, I tried every number up to 4000 to ensure that everything before that was handled correctly.