Evan Nichols
EECS 678
Lab 2 Report

1. What is wrong with the original code that eventually causes it to crash? (Obviously, the program fails becuase of an invalid memory reference producing a segmentation fault -- what programming error led to the seg fault?)

The program crashed due to an error saving the_printed_command in the execute_cmd.c file on line 3672. It was attempting to free an already freed block argument. The buggy lines of the code were:

```
FREE(the_printed_command_except_trap);
the_printed_command_except_trap = the_printed_command;
```

When comparing this section of code to similar calls in other parts of the program, it was clear that the *savestring()* function must be used when saving the the_printed_command. To fix the bug I added:

```
the_printed_command_except_trap = savestring(the_printed_command);
```

After re-compiling, the program ran as expected.

2. Describe how you diagnosed the problem with the original code. If you used GDB, which commands did you find most helpful? If you did not, what tools were most helpful in diagnosing the problem?

I diagnosed the problem using GDB. Setting a breakpoint on the execute_command function was very helpful, and then using "list" to inspect the code around the breakpoint. After, I used "fin" to continue to the end of the function, which then showed me exact line of the error.

3. Describe how your solution fixes the problem. Are you confident your solution is correct?

Yes. Instead of simply assigning the_printed_command_except_trap to the string the_printed_command, utilizing the savestring() command actually copies the_printed_command into the allocated memory being pointed to by the_printed_command_except_trap. It is a classic memory leak - you get pointer to some allocated memory, then lose the only reference to it when you assign the pointer to the string literal the_printed_command.