CS426 Project 1 Warm-up

System used: I used SSH to run my code on data.cs.purdue.edu

1. Problem 1
    a. For each of the scenarios, I think that an error will be displayed as each of the indices are not in the range of the array. Accessing indices outside the bounds of the array should result in undefined behavior. I also think that invalid memory addresses will be printed out for each negative index.
    b. I attempted to store a value at each index, then print out that value and its address. When I printed them, each index showed the correct value that I previously stored. Also, valid memory addresses were printed out when comparing them to the starting address of the array. My predictions were not correct because I was able to successfully access indices outside the bounds of the array and they each had a valid memory address.
    c. The three other values I used were -30, -40, and -100000. I was successfully able to store a value in each of these indices. When I printed the element at these indices, the correct values that were previously stored were printed out as well as valid memory addresses.
2. Problem 2
    a. For each of the scenarios, I think that I will be able to successfully store values at the specified index and those same values will show up when printing out the element at each index. Also, I think that valid memory addresses will be printed out for each negative index.
    b. I attempted to store a value at each index, then print out that value. When I printed them, each index showed the correct value that I previously stored as well as a valid memory address when compared to the starting address of the array. However, when trying to store a value at -10000, a segmentation fault occurred indicating an invalid memory address. Therefore, my predictions were not correct because I could not successfully store and print an element for the index -10000.
    c. The three other values I used were -30, -40, and -100000. I was successfully able to store and print the correct values and valid memory addresses for the -30 and -40 indices. However, another segmentation fault occurred when trying to store a value at -100000.
3. Problem 3
    a. In many cases, using negative indices in an array will result in a segmentation fault. When a segmentation fault occurs, there is a possibility that a core dump will also occur. Normally, core dumps are used to debug programs, but if an attacker were force a segmentation fault resulting in a core dump, they could gain access to sensitive information, such as passwords, through the file that the

program's memory is saved to. However, when a segmentation fault does not occur, memory that belongs to other parts of the program or other program(s) is overwritten. Attackers can use this knowledge to overwrite specific memory with whatever they choose, such as malware.

b. A common situation when an off-by-one error occurs is when using a loop (specifically a for loop). When using a for loop, it is necessary to input the correct bounds of the array to ensure that the loop does not iterate to an index outside of the array's range. To ensure this does not happen, we can use a for-each loop. By using a for-each loop, the loop will only access elements that exist inside the array's bounds. Also, if using a for-each loop is not an option and a for loop must be used, ensure that the starting iteration index is always positive and the ending iteration index is always less than or equal to the length of the array minus 1. When declaring the ending iteration, it is important to create a variable that is equal to the length of the array minus 1 to make sure there is no disconnect between the size of the array and the number of iterations the for loop will perform.
c. Memory leaks might result in a security problem such as a denial of service. If an attacker forces a memory leak, it is possible that the allocated memory may be left in a state where the system does not know that it should be reallocated. As more memory is left in this state, the software will run much slower or crash altogether. A use after free might result in a security violation by an attacker injecting code into a program. If a program refers to a piece of memory that has already been freed, it will result in a dangling pointer. An attacker could then enter their own data to allocate to the same piece of memory which is referenced by that dangling pointer. This allows attackers to substitute code for whatever they want such as malware. Lastly, uninitialized variables might result in a security violation due to the stack not initializing variables by default. If a variable is not initialized, it most likely contains some data from the stack. An attacker has the potential to read this data which can result in manipulation of the stack and/or visibility of sensitive information.