Noah Vienneau

CS 0449 Project 2 Report

File One – Answer = "EaKOVfvBdNYcuBDgprqQgQNmpX":

For file one I used gdb and set a breakpoint at main so I could disassemble the program and see what functions where being called in the program. I set a break point at this a cmpsb statement which I assume was comparing between the $esi and $edi registers to see if the strings matched. I then used x/s on both the $edi and $esi registers and saw that $edi contained the string I entered, which was “Hello” and the $esi register contained this : "EaKOVfvBdNYcuBDgprqQgQNmpX", so naturally I ran the program again only this time with the string contained in the $esi register as input, and I received the message telling me I was correct.

File 2 – Answer = “nmv28”

A bit embarrassing to be completely honest as I didn’t realize the fact that the two strings being compared in the strcmp function where my input and the string nmv28_2, stored in the $ebx and $esi registers respectively. I also completely forgot that $esp is the stack pointer, for a while, until I went back to the GDB puzzle lab to look over how I got my solution for that. The key to this program was understanding that a _2 was being added to the end of every string I inputted, and was then being compared to nmv28_2. I realized this by inputting nmv28_2 and right before it was being compared it had changed to nmv28_2_2. This realization allowed my to come to the conclusion that by inputting my pitt username it would automatically have the _2 added to it; thus, the strcmp function would return true and the program would be solved.

File 3 – Answer = Any 9 letter string that contains 8 capital Z, i.e ZZZZZZZZj or ZhZZZZZZZ

Trying to enter a breakpoint at main in gdb I noticed that there was no actual main in the info function table call, so I set a breakpoint a 0x0 and called the info functions and saw getchar, which made logical sense to set a breakpoint at since string input was involved with the program.

After stepping through the getchar function I got to a section that I had to identify using objdump to realize it was part of the .text section as being part of the text section. I then noticed that that it loops 9 times through getchar, as it took 9 enters of continue in gdb to get too the end of the program. I then noticed a call that put the %al register, which after reading online found out it stored an inputed character into the into the $eax register, and is then subtracted by 0x58. It then sees if the value is equal to 0x2, if it is it increases a counter in the location -10($ebp) by 1. Afterwards there is a cmpl instruction that compares the value 0x8 to

the value stored in -10($ebp). This means that the character above mentioned character must be inputted at least 8 times into the answer.

By converting 0x58 to decimal, I got 28 which subtracted by 2 equals 26; consulting the ascii character table I saw that a capital Z was the character that corresponds to 26 and when entering at least 8 Z into the input I was able to unlock the puzzle.