Sai Konduru

Professor Ahn

Cs 449

22 October 2016

Project 2: Passwords

1. Sck42\_1. I started off setting a break point at main and then used disas to dump the assembler code. Looking through the code I saw a line of code that caught my attention, Rep cmpsb. It looked like it was a compare function so I looked up what it does and sure enough it is a compare function. From that point I knew that one of the two arguments had to be the password and the other had to be what I entered. I understood from looking at the code above that register $edi contained what I input because this was the same register that was passed into fgets and chomp functions. So register $esi has to have the password. When I used the input x/s $esi I got the string “haTfPtGSaOpdIlpsiJKj” and when I input this in as the password the program returned “Congratulations!”.
2. Sck42\_2. I started off setting a break at main and disas like before. I saw there were 3 function calls to function name I didn’t recognize. The functions were c, p, and s. So I set break points at these functions and used disas to see what’s happening in the functions. I started with c and I found that this function removes the enter character or the \n character. I found this by putting a break before and after the function call and saw what happened to the register that got passed into the function. Next I looked at function s and I found that this function counts the number of characters. I saw that in s there was a counter variable and it looped. I assumed that the function was character count. To prove my assumption, I set breakpoints before and after and saw that eax was the number of characters in input. Finally, I moved to function p. I saw that in main function p has to return 1 for the password to be successful or to get the congrats message because after function p there was a cmp that compared 1 to eax and jumped to the sorry message if they were not equal. Taking a close look at function p I found that goal was to compare that first and the last character. Register edi was set to the last character of input because I set a break after lea -0x1(%esi, %eax, 1), %edi and did x/s $edi to find that edi had the last character of input. Register al was set to the first character of input because I set a break after the line cmp (%edi), %al and did x/s $al to find that al was the first character. Therefore, cmp line compares the first character of input to the last character of input and sets eax to 0 if they weren’t equal. Then I noticed that there was a loop and some changes to what seemed like counter variables. These counter variables were edi and ebx. The sub $0x1, %edi line lead me to believe that edi is moving closer to the center. Meaning after each increment in the loop edi will hold a character closer to the center starting with the last character. The very next line was incrementing ebx by 1. This lead me to believe that incrementing ebx was moving al closer to the center and after checking this theory with a break after the first time I saw al, I found that al is the next character in input. Therefore, the function compares each character moving inward. In other words, p was looking for a palindrome. However, not every palindrome works because later in main, s is called on input and the return is compared with 0xd. This means that the length of the input has to be greater than 13. So the password is any palindrome with length greater than 13. Worked with ABCDEFGHGFEDCBA, ABCDEFGGFEDCBA and AMANAPLANACANALPANAMA.
3. Sck42\_3. I started off attempting to put a break at main but found that the program doesn’t have a main. To dissemble the program used objdump -D sck42\_3 and was able to get the dissembled version of the program. With prior knowledge of assembly, I knew that .text has to be the new main. At first I was looking into the library functions to get an understand of what’s happening. I soon found that this would result in unnecessary work because I realized that the library function was just getting a character using the getChar function. So I continued through the .text part of the code and I saw some cmp functions that seems to lead to the congrats and sorry prints. From the previous project I knew that these cmp functions are crucial for determining what is the right password. One of such cmp functions was the line cmpl $0x9,-0xc(%ebp). Here it is comparing the immediate value of 9 to what ebp points to offset by -0xc. I was able to figure out that this is a counter for a loop. Within the loop we count the number of characters until we hit 10 characters counted. The characters counted would be 10 because we start of setting -0xc(%ebp) to 0 and count until it hits 9, (9-0) + 1 = 10. Therefore, no matter what you enter the function only looks at the first 10 characters. Another crucial cmp line is cmp $0x4, %eax. The line before was sub $0x31, %eax. Earlier in the code we saw that %eax was set to the current character that the program is looking at—figured this out by putting breakpoints at various areas and determining that eax is equal to the ascii value of the current character. According to these two line, after our current character is subtracted by 0x31 the value has to be equal to or less than 0x4. This establishes a range for which the characters must fall under. Adding 0x4 to the 0x31 gives us 0x35. So the range would be 0x31 to 0x35 inclusive. Looking at the ascii table we can see that 0x31 is 1 and 0x35 is 5. So inputs have to be 1-5 inclusive. The ja after the cmp is how I was able to find the range. If we trace what happens when the ja executes, we can see that it skips an add operator, addl $0x1, -0x10(%esp). Following what happens if this add does not execute I realized that eventually it will jump to the sorry message, meaning I would get the wrong password. This lead me to believe that this ja should not execute and for that to happen the inputs have to be from 1 to 5. Stepping through the code and looking at the previous compare, I was able find that -0x10(%ebp) is actually a counter for number of correct inputs because it adds 1 to -0x10(%ebp) every time we don’t execute ja from above or every time it meets the requirement of being within the range of 1 to 5. There is one more important compare at the end that would cmp 0xa, -0x10(%ebp). Therefore, the correct password is any combination of 1-5 characters in the first 10 characters of the input. Worked with 1234554321, 1234512345kjsldfjks, 4444444444 and etc. One note to remember is for my specific program multiple line entries would fail because I require 10 out of first 10 characters to be the accepted values but having an enter key within these 10 makes the condition false. The condition would fail because the program counts the enter key as a character.