Cs449 Project 2 Write-up

Craig Mazzotta

[Cam292@pitt.edu](mailto:Cam292@pitt.edu)

**Cam292\_1**

**Procedure**:

My first attempt at cracking the password was running the mystrings program from part one on cam292\_1 to see what I could find. After searching through the output, if found the string ”pXAdWqntXtwpROQPyfWAo” on the lines directly before “Sorry! Not Correct!” and “Congratulations!”, so I figured that would be what the user input would be compared to. I started the program, entered the string I found and it worked!

**Solution**: pXAdWqntXtwpROQPyfWAo

**Cam292\_2**

**Procedure:**

To begin, I placed a breakpoint at main and disassembled the code. I saw a call to a function c, so I placed a breakpoint at it and disassembled the code again. I saw, then, that c made a call to a function s, so I placed a breakpoint and disassembled. Stepping through s, I found that it incremented %eax for each character in %edx, where %edx is the user entered string + a new line character. In doing so, I found that s acts like strlen(). At the end of c, I notice that the new line character has been stripped from %edx, then it returns to main. Continuing, I see a call to a function p and another to s. Directly after s, I see that %eax is being compared to 0xa followed by a less than or equal comparison, so I came to the conclusion that the password must be at least 11 characters. Next, I placed a breakpoint in main at p, then disassembled the code. Stepping through, I found that p was taking the string entered, reversing it, and checking if it was equal to the original.

**Solution:** any string whose length is greater than 10, and must be the same forwards and backwards. Ex: aaaaaaaaaaa or 5aaaaaaaaaa5

**Cam292\_3**

**Procedure:**

Originally, I ran the program with gdb and tried playing a breakpoint at main. I found out there was no main label, so I ran ‘objdump –D cam292\_3’ and found a function call to label ‘\_\_libc\_start\_main’ in the .text section at address 0x080483bc. Since it was the first function, I believed it could be main. So, I ran it with gdb again placed a breakpoint at ‘\_\_libc\_start\_main’. Stepping through the program, I found a call to a function on address 0x00bd3d23 that did not have a label. I found this to be the actual main function.

Stepping through main, there was a call to getchar at address 0x0804846e. It begins by prompting for the password, then stores the ASCII value of the character in %eax. After investigating further, I see that the call to getchar is in a loop in order to get the length of the string entered. This value is being stored at -0xc(%ebp). Once the length read in is 10, the program exits the loop, stores 10 in %eax, then jumps to 0x80484b9. From this, I can see that it only cares about the first 10 characters entered, and if the input is less than 10 chars, it prompts the user for the additional characters.

At 0x80484b9, the main function enters another loop. This time, each character is passed to the function tolower and the corresponding lowercase ASCII value is stored in %eax. After exiting tolower, 49 (0x31) is subtracted from %eax, which is then compared to 4. If %eax is not equal to 4, the counter at -0x10(%ebp) is incremented. From this, I came to the conclusion that the character ‘5’ (ASCII value 49+4) is a special character. Otherwise, one is added to the current character count and the loop begins again. Once the loop runs 10 times, the count of character ‘5’ occurrences is compared to 7. If the count of occurrences is not equal to 7, the password is then incorrect.

**Solution:**

Password must be 10 characters or more, containing at exactly 7 occurrences of the character ‘5’ in the first 10 characters of the string.

Possible solutions: 5555555aaa, a5a5a5555555