Mohit Jain (moj10)

CS 0449

March 1, 2017

Password Project

Program 1

For this program, the way I went about solving it was that I first decided to run the program. After I ran the program I typed up a really arbitrary password, which obviously did not work, but after I typed up this password I noticed that this file was a program when it printed out “Sorry! Not correct!” At this point, I realized that there must be a function that tells the prompt to print out such words if the user does not get the correct password. Obviously, this function was easy to tell. It was a printf.

Instead of going through the process of using disas and searching the addresses for each part, I just navigated to the point in the file where the “Sorry! Not correct!” was called and looked around in that area to find a passphrase using the string – all filename | less command on thoth. This saved time and was very efficient as I actually did find the passphrase one line above the “Sorry!” line. **The passphrase was “yCkgASQwZEMhRgTyZqwqsNj”.**

Program 2

For this program, the way I went about solving it was that unlike the first program, I just could not easily find it by searching the file for a passphrase around the area. Instead what I had to do was from the start I needed to do disas /m main and completely read through the assembly file that I got from it and find out what it was doing. Throughout reading, I found out that a lot of the functions had to do with the time. These functions were, in order of placement, time, localtime, and strftime. In order to find out what these functions do and print out I read through the code and learned that after the time functions, I noticed that they were taking out the first letters of every word, and found something really interesting in that.

I then continued on and got rather annoyed. At this point, it was more or less guess and check. I started picking addresses towards the end of each function call and tried printing the strings of each of the six registers we have available. In one of the registers, $ebx, I found the very first word “Wednesday.” In a second register, I found the word “March.” At this point given that the program was taking the first letter of each word out, I just assumed that this could lead to the passphrase, so I tried two possibilities: WM and MW. The MW passphrase did not work, but WM did. **So overall, the password for program 2 was “WM”.**

Program 3

As for program 3, I was unable to reach a solution for a passphrase. To describe what I was trying to do, I first started out by running the program on gdb and I entered a one character passcode on my first run. Then, something very different happened. I was able to enter up to 5 one character passcodes. I tried this for 2, 3, 4 and so on till I got 9 character passcodes and discovered that after a passcode the 8 character limit, I could not enter in another passcode. So in such a case, I would only be able to enter one 9 character passcode, or two 8 character passcodes. After my many trials of running the program, I then went on to run the program on my gdb. When I first ran this program, I learned that I was unable to run this program from main as no main existed and even when I was trying to disassemble the functions, in which I started from the getchar function for which I found about after I ran the file with the strings command, I just could not find any sign of a main function. There was no symbol table. I was literally working and trying to find a passcode with almost nothing at hand.

Regardless, I did not give up. Instead of starting at the first function I saw, a puts() function, I started down below with the getchar() function and tried to back trace my way up hoping that I could early on find the passphrase. However, this did not work as I was just looking at infinite amounts of garbage assembly code. I actually tried to print out some of the register by using the print command in gdb and the x/s command in gdb. I never got anything useful. I did this for about 3 to 4 hours and nothing came up. At max, I ended up seeing this weird loop and some kind of bitmask.

Overall, the only thing I found useful about the passcode was either that there were going to be multiple passcodes that were each less than 8 characters long or there could have only been one passcode that would need to be at least 9 characters long.