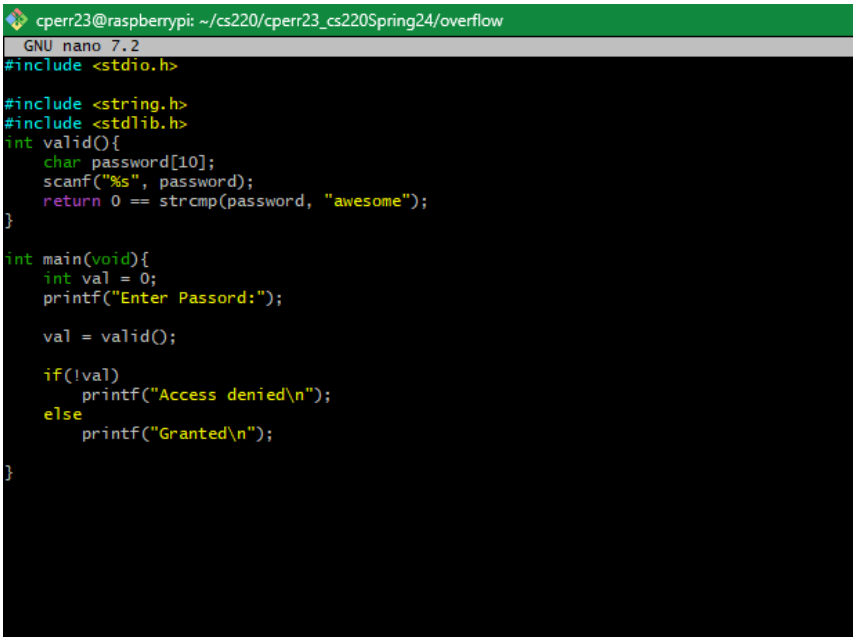
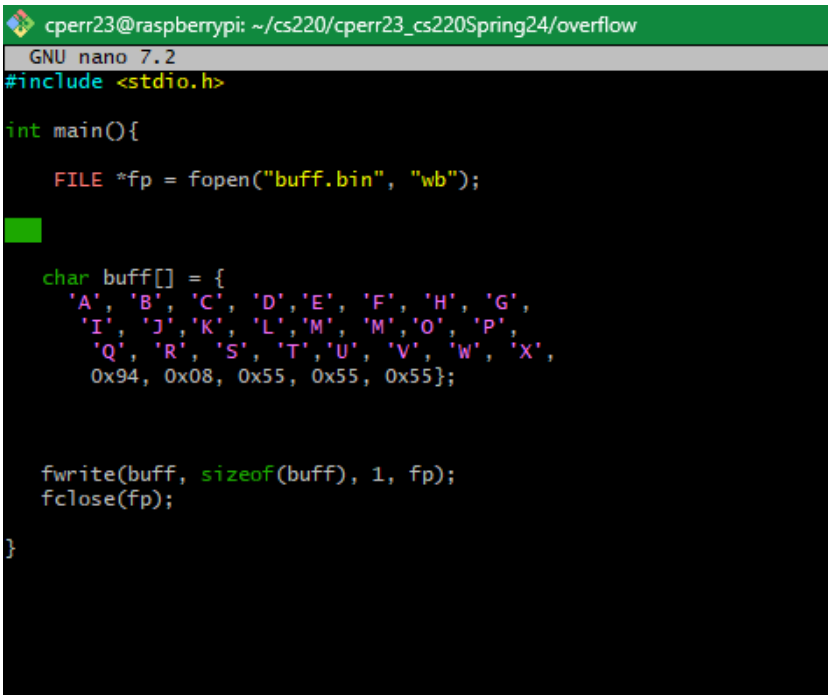
Reference:

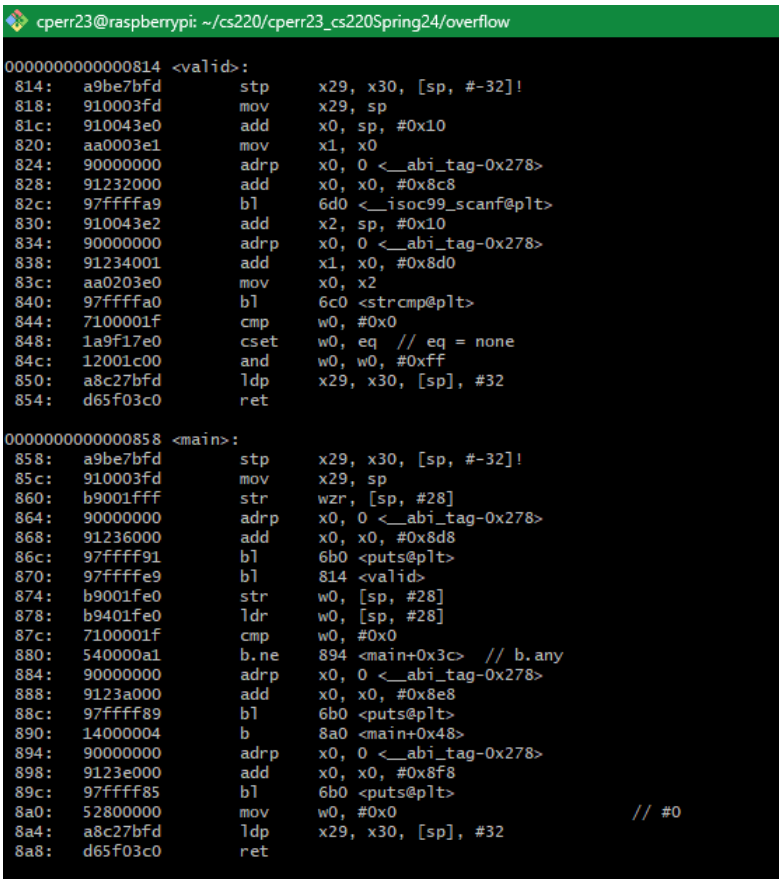
Here is the C code that I am testing the overflow on



Here is the code that I use to create the character array to overflow the buffer

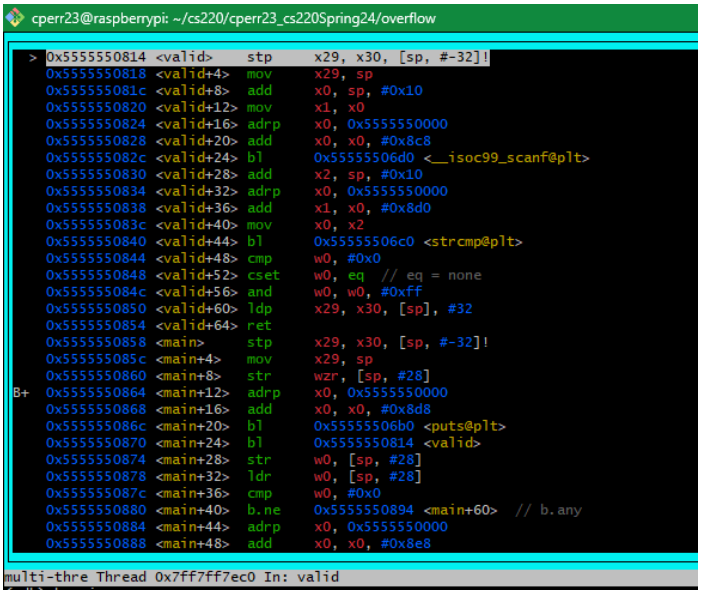


Here is the object dump of the C code where I find the address to overflow

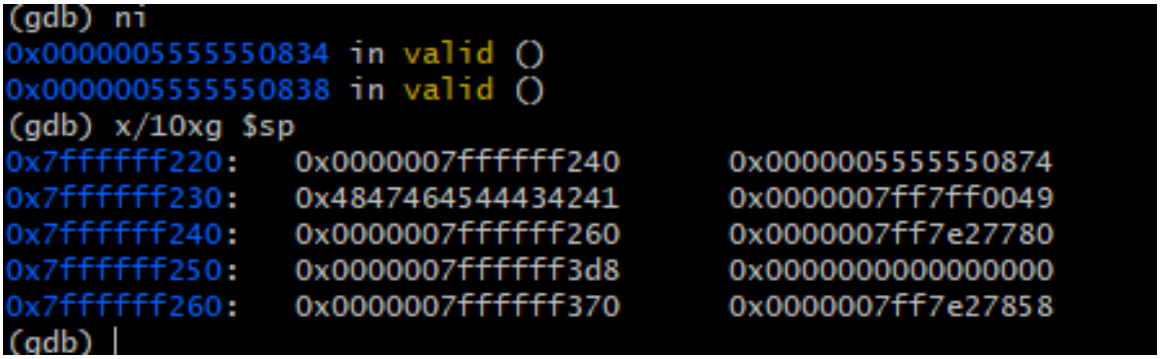


To begin with I knew that the character array was 10 bytes long, meaning to even get out of the password I would have to start with at least 10 bytes, thus that is the reason to start with random letters.

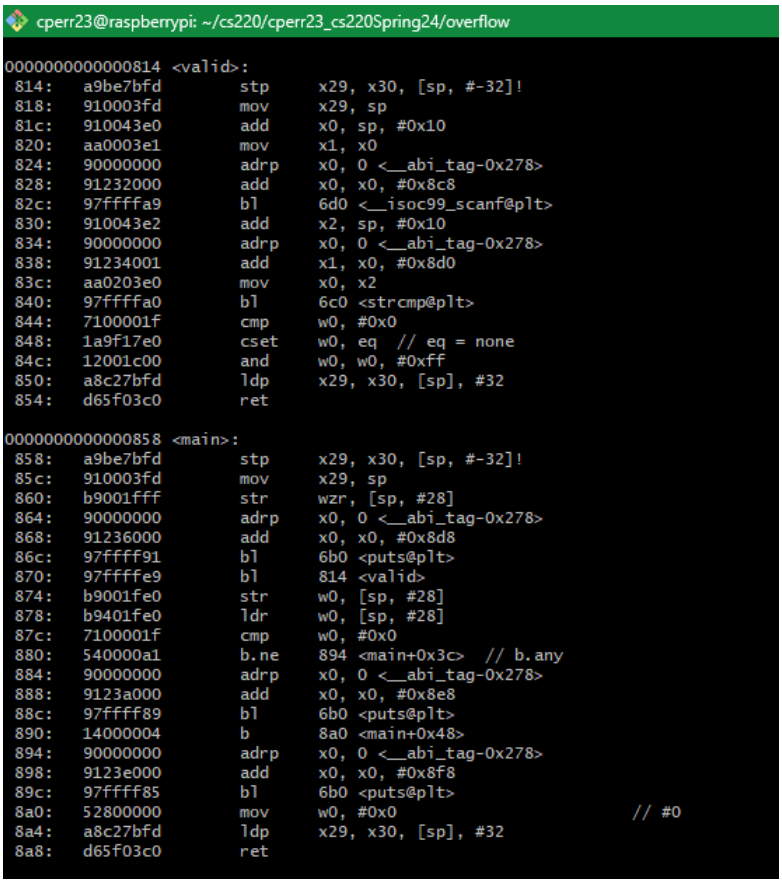
To start the debugging process I create a break point at main or b *0x555550858, once I am in main I keep stepping through until we get to the valid function, then I step into the valid function



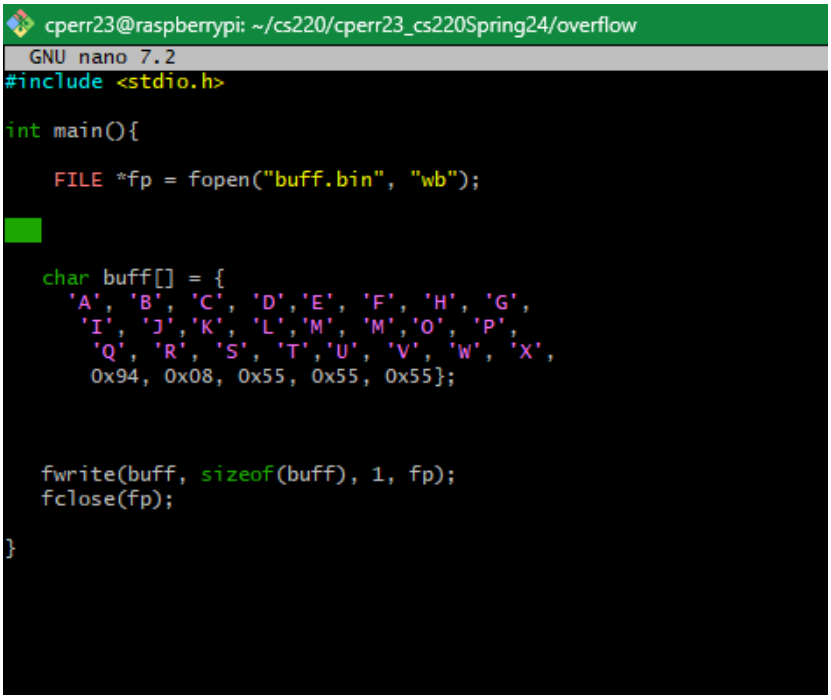
Once we are in the valid function then we can keep iterating through instructions until we hide the scanf function call. Once we are there I decided to enter ABCDEFGHI, once I had entered that to the scanf function call and it returned I checked the stack to see if it had been allocated correctly.



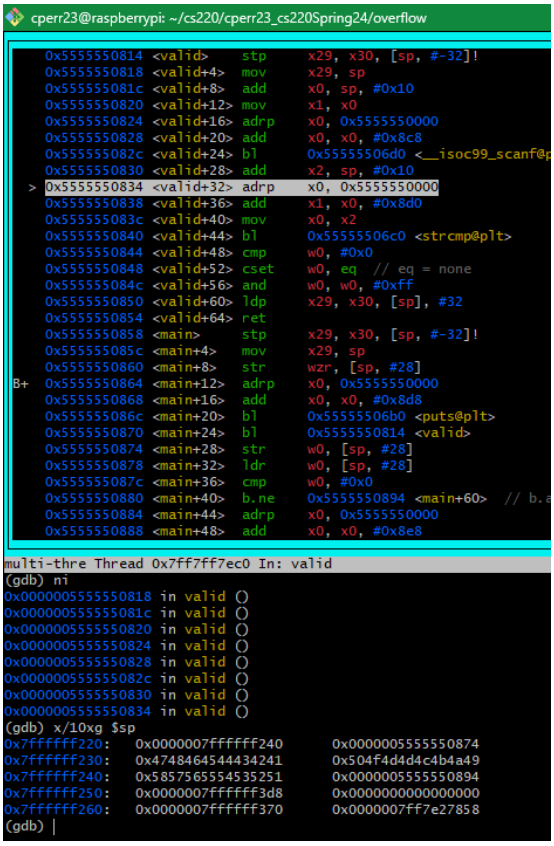
As you can see with 0x7fffffff230 we can see A as an ascii 41, then B as 42 and so forth. We can also see the frame pointer and link register on the address above at 220, thus we know that if we clobber the link register and make it an address for “Granted” then we can backdoor into the program without the password, thus we need to look at the object dump



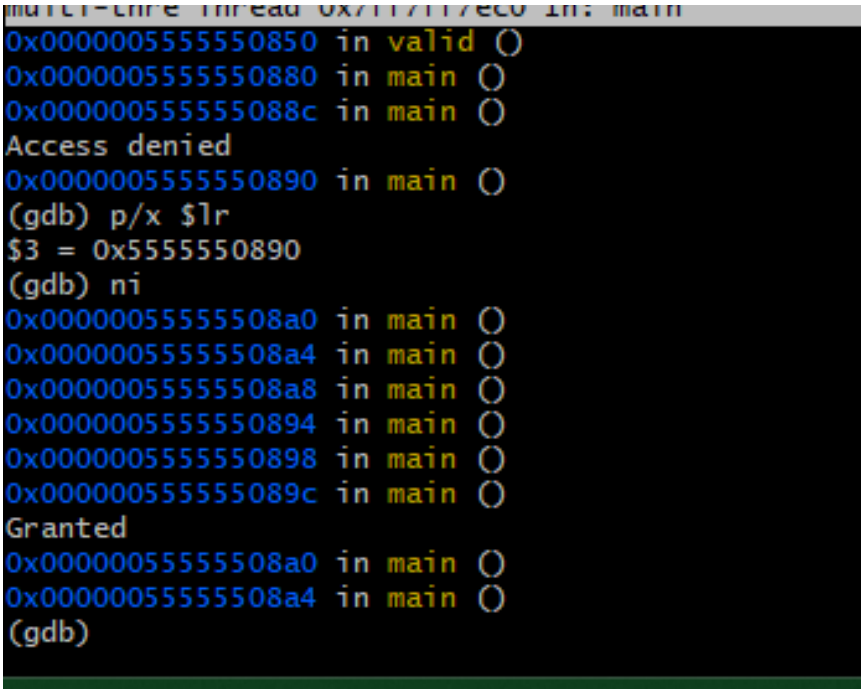
We see with the C code that if the comparison is not equal then it is “successful” thus looking at b.ne to 894 then we know that is the address we need to insert into the program. So, we have the address 0x555555094 and if we enter 10 bytes to get rid of the buffer, then 6 more to complete the row of array of memory, use 8 more bytes to clobber the frame pointer we can then use the next bytes to enter an address which will replace the link register. This is where the WB program comes in, as we cannot enter character values themself as a ‘5’ isn't a 5 in ascii. Then we need a program that we can fill with junk then at the end the correct address and with all that will convert it to binary that we can directly insert into the scanf program, thus we have the wb.c file



We can fill the array with junk then add the address, in little endian format, at the end and this will write to a binary file that we can insert with the ./function < buff.bin. Which is exactly what we do. It does crash the system but if we step through with gdb we can see it grants access.



We can see the 0x5555550894 and if we continue it will say acess denied at first but since we changed the link register for the main output it will brute force into “Granted”



Now we have gotten in by overflowing the buffer and rewriting the link register!