**William Kurek**

**Project 2**

**Passwords**

**PROGRAM 1:** wrk10\_1

**PASSPHRASE:** FvpbNgVVHSdIjZkFwufglLpWcfv

Using the mystrings program I created, I printed out the strings encountered in the executable of the first program. The first things the caught my eye was the set of strings:

**Sorry! Not correct!**

**Congratulations!**

**Unlocked with passphrase %s**

I then quickly realized that the passcode could potentially be in the same area as these strings. That is when I noticed a unique string, different from all the other ones printed, directly above the string above. The string was:

**FvpbNgVVHSdIjZkFwufglLpWcfv**

Immediately after seeing this string, without attempting any other methods, I used it to unlock the program with ease.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**PROGRAM 2:** wrk10\_2

**PASSPHRASE:** billy, wrk10, or whatever username was used to execute the file

First, I attempted to once again use the mystrings program to find the passcode, but I knew it wouldn’t be that easy. I was right. Without any luck with mystrings, I decided to disassemble the program in GDB using the command “disass main”. From here, I searched and analyzed the assembly code looking for any clues to where the passcode could be stored. That is when I noticed a reference to the C function “strcmp” in the assembly code. I figured the password would have to be in this area. Therefore, I set a breakpoint to right before the execution of the “strcmp” function, where I noticed a register value was being passed to another register:

**mov %esi,0x4(%esp)**

**mov %ebx,(%esp)**

**call 80483d8 <strcmp@plt>**

I quickly assumed this had to be the passcode. I then ran the program and once the breakpoint was reached, I printed what was in the “esi” register using the command “x/s register $esi”. Being a string that was printed, intriguingly my username, I used it to unlock the program, and it worked. I then realized the password changed as the username, for I’m assuming terminal, changed. So the passcode is the username at that point in time.

**PROGRAM 3:** wrk10\_3

**PASSPHRASE:** %%%%%%xxxx, //////xxxx, %x/x%x/x%/, and more…

Unlike the first two programs, the third gave me a significant amount of trouble. With the methods I used in the first two programs not working, as I was unable to disassemble the program in GDB, I hit a road block. This file was strange. The debugger was unable to find a main function, and the breakpoints I set led nowhere. The only important piece of information I received from GDB was that the string was 10 characters long, as the toLower function was called 10 times regardless if 10 or more characters were entered. I decided to take a day break or so on this program, to refresh my brain. When I continued the following day, I decided I had to think outside the box on this program, as the passcode wasn’t going to be in hind sight like the previous two. So, I decided to use “Objdump -d” to disassemble the program. With the assembly language finally in front of me, I thought the road ahead would be easy from this point on. I was wrong. I realized that I really had to dissect the assembly code to find what I was looking for. There were no signs of two strings being compared and no clear loops or segments of a main function. But, thens when a particular segment of code caught my eye:

**\*sub    $0x25,%eax** // subtract 0x25, or “%” **\*cmp    $0xa,%eax //**compare to 0xa, or “/” after compare **ja     80484cb <tolower@plt+0x143> //**jump if above **mov    $0x1,%edx  
mov    %edx,%ebx  
mov    %eax,%ecx  
shl    %cl,%ebx  
mov    %ebx,%eax  
and    $0x561,%eax  
test   %eax,%eax**

**je     80484cb <tolower@plt+0x143>  
\*addl   $0x1,-0x10(%ebp) //** increment count by 1 **nop  
addl   $0x1,-0xc(%ebp)** //next char **cmpl $0xa,-0xc(%ebp)  
jle    8048492 <tolower@plt+0x10a>  
\*cmpl   $0x6,-0x10(%ebp)** //check if counter = 6 **jne    80484f1 <tolower@plt+0x169>**

I quickly realized this loop compared a particular register to various values through subs, cmps, shifts, movs, and loops. This loop detects if a set of specific characters were entered EXACTLY 6 times after iterating through 10 characters. At this point, I knew I was getting close. So I converted 0x25 and 0x25+0xa into their respective ASCII values. These two values were “%” and “/”. So, I determined these specific values must be used EXACTLY 6 times to unlock the program. To my relief, my calculations were correct as various combinations of those values EXACTLY 6 times unlocked the program.