Randyll Bearer: Proj2\_Password Write Up

Rlb97\_1:

Password: eXxjVoAEsgoemITbSTLBy

To begin, I placed a breakpoint at main and ran the “disas” command. I highlighted the four function calls and placed breakpoints at both <fgets> and <chomp>. From here, using “x/s” I was able to see that the register %edi held my input string “abcdefg” and was being put onto the stack before chomp was ran, but didn't see it used afterwards. At instruction addresses 0x08048302 and 0x0804307 I saw multiple registers being loaded, including %esi. Then I came upon the “repz cmpsb” instruction, which I had to search online for. I found that it is an instruction which compares strings, and that it was comparing %edi (my input string) and %esi (a register I saw a value be moved into). I placed a breakpoint here and used “x/s” on %edi, it was still holding my input, and did the same for %esi and found a random string (eXxjVoAEsgoemITbSTLBy) which I assumed to be the password. It was, and over multiple runs it continued to work which gave the impression that the password for this program is static.

Rlb97\_2

Password: Condition 1: Password must be at least 12 characters long

Condition 2: Let x = the floor of └ input length/2 ┘. The first x characters of the password must be symmetrical to the last x characters.

Examples: “zzzzzzzzzzzzz” - 13 z's. ACCEPTED

“zzzzzzzzzzzz” - 12 z's. NOT ACCEPTED

“1234567654321” - x = 6. First x = “123456”, Last x = “654321”. ACCEPTED

“12345678654321” - x = 7. First x = “1234567”, Last x = “8654321”. NOT ACCEPTED

Initially I set breakpoints at all the function calls <c>, <p>, and <s>. I noticed a looping pattern of c-s-s-s-p-s-s, but did not notice anything as obvious as in rlb97\_1 so I decided to trace/translate the entire program. After tracing I noticed two key cmp/jmp instructions at 0x0804852c (jump to fail state if %eax != 1) and 0x0804853d (jump to fail state if %eax <= 12). If either of these conditions are met then the password will not be accepted, so I knew that I must then define the opposite.

- For line 0x0804853d we need the function call to <s> to return a number > 12 in %eax

- For line 0x0804852c we need the function call to <p> to return 1 in %eax.

As for what I have determined these functions to do:

<c>: Eliminates the terminating character '/0' at the end of input

<s>: Returns %eax with input length

<p>: Checks the second condition listed above and returns 1 in %eax if true.

I struggled for a little while trying to figure out what <s> accomplished, but struggled a whole lot more trying to figure out <p>. At first I thought that it had solely to do with the five instructions on 0x080484d9 – 0x080484e3, since if the condition for the jl (0x080484e3) was not met, then we could move directly to the “mv $0x1, $eax” instruction which is what we need <p> to return with.

0x080484d9 <+62>: shr $0x1f,%edx //%edx holds input length

0x080484dc <+65>: lea (%edx,%eax,1),%eax //if leftest bit = 0, just eax, if 1 = eax + 1

0x080484df <+68>: sar %eax //shifts to the right 1 time preserving sign bit

0x080484e1 <+70>: cmp %eax,%ebx //(ebx-eax)

0x080484e3 <+72>: jl 0x80484ba <p+31>

**0x080484e5 <+74>: mov $0x1,%eax //This is what we need**

I was convinced for a while that the trick here was to have %eax (holding input length) to have a 1 bit in the leftmost position, so that after the sar instruction it would be shifted to the right and filled with another 1. Since this would then cause %eax to be interpreted as a negative signed value in the cmp instruction, it would be able to bypass the jl instruction. However, I realized that that would require the input to be far greater than 100, thus breaking the specifications of the project.

Once I realized that %edi was decreasing (thus going backwards through the end of the input) and %ebx was increasing (going forward through the beginning of the input), I started to pick up on the trend that the program was checking symmetry. Once I found this out, in conjunction with the cmp's after <p> and <s>, I could decipher the above conditions that are required for a valid password

Rlb97\_3

Password?: Condition 1: Must contain at least 10 characters

Condition 2: Newline also counts as a character

Condition 3: Must contain at least 5 numbers

First thing I noticed about rlb97\_3 is that the program accepts various amounts input strings. To receive output it took five single-character strings, three three-character strings, two six-character strings, and one twenty-character string (“12345678901029384856”), which happened to be accepted as “1234567890” so I guess I got lucky. I further tested inputs and found that the maximum accepted input length was 10, that the program requires the input to include at least 5 numbers, and that newlines count towards the 10 character minimum. For example,”\n\n\n\n12345\n” would be accepted by the program. I also noticed that sometimes duplicate numbers would be allowed and other times they wouldn’t be, so I had to check the code for a more concise answer.

I tried placing a breakpoint at main and wasn't allowed by gdb, so initially I was at a bit of a loss. I tried placing breakpoints at other functions which appeared in other programs (chomp, fgets, c,s,p) and stumbled upon printf and puts, which both worked allowing me to disas the code. I placed breakpoints at the function calls “\_\_i686.get\_pc\_thunk.bx”, “\_\_strlen\_ia32”, “\_L\_unlock\_232”, and “\_Unwind\_Resume” only to get input/output errors, but I was able to disassemble them at least.

I tried tracing through this program, but I was not able to use GDB effectively this time around. Setting breakpoints on addresses would give me errors, registers would be considered out of bounds, and other problems stopped me from being able to accurately trace rlb97\_3. I understand that this is not a fully acceptable answer, but I would like to at least submit a couple hypotheses in addition to the previous 3 conditions.

1. I believe that number duplicity is relevant to correct passwords, as “1234567rlb” is accepted but not “1111222rlb”, even though “rlb121rl12” *is*. Clearly adjacent duplicates play a part
2. The program only cares about the first 10 inputs of the password. It can accept *more*, but it always truncates it to 10. This is given away by the “unlocked with passphrase” line which only feeds back your input up to 10 characters.