**Buck Young -- bcy3**

**Passwords Write-up**

**bcy3\_1** : Unlocked with fNFCklDRjBGBwKSuWiOPiq

This program was very similar to the recitation example. First I explored the chomp function without finding anything of interest. So I returned to main and discovered that the program accepted a string, called chomp, and then compared my string with a set value. In this case, the interesting lines were:

0x08048302 <+51>: mov $0x80b316c,%esi

0x08048307 <+56>: mov $0x17,%ecx

0x0804830c <+61>: repz cmpsb %es:(%edi),%ds:(%esi)

The correct string is moved to $esi in 0x…302. My string is in $edi! After a bit of research on the repz command there, the solution was obvious.

**bcy3\_2** : Unlocked with FO, WO, MO, [etc]

The password is the first letter of the day of the week and the first letter of the current month. So, this month on Friday, the password is “FO” and on Monday it is “MO”.

Initially, I assumed that this program was just as simple as the first. I set a breakpoint before the string compare, examined a few variables, found “FO”, and called it a day. However, the next day my password didn’t work. What’s going on here? I watched for changes in $edi as I walked through the program and found that main is calling <localtime> and <strftime> and building a password from them. Bingo!

**bcy3\_3** : Unlocked with buck buck, lizz lizz, bcy3 bcy3, aaaxxxxaaa [and more!]

The password is ten characters long and requires exactly four “special” characters. It can be any combination of any symbols or letters as long as it contains exactly four of the following: b, c, m, n, v, x, z

This program took me *days* to solve. There are so many dynamically loaded libraries, no main function, no chomp function, and furthermore so many passwords seemed to work! And they all seemed to be connected to me in some way – my name worked, my pitt username worked, my girlfriend’s name worked, what the hell was going on?!

After a few days of banging my head against the wall and after a few crack-pot theories about why the passwords were connected to me (can this program access my pitt mail?!), I felt like I had made some progress.

A lot of strings were being read in during loading from various places and something called memset() seemed very important. I had a theory that the program was putting a bunch of strings in memory from the shared-object files and somehow comparing against those. For instance, these strings were being read in:

“FCEDIT=emacs”, "LOGNAME=bcy3", “/lib/libc.so.6”

And guess what? “Emacs Emacs” worked, “bcy3 bcy3” worked, and “libc libc” worked!

But I couldn’t find where the comparison was happening. It never seemed to compare my input with *any* string. What the hell was going on?

0x80484bd: and $0x1501803,%eax

This line seemed absurd and eventually became crucial in understanding how the program functions. So what I did was find a decent breakpoint just before the unnamed (“??”) function which I was interested in (break \_dl\_runtime\_resolve). Then, since I couldn’t disassemble the code, I displayed 30 instructions after $pc. Bingo! I had a nice list of what was happening in the “??” function!

What I found was that the program, in simple terms, gets 10 single characters and subtracts 0x62 from them and compares the result to 0x18. Then, it does a jump based on the unsigned value. If the jump isn’t taken, some things get moved around, some bits get shifted, the logical AND above happens, and eventually (if another jump is not taken) a counter is incremented. This means that only a small subset of ASCII characters make it though this function-obstacle-course to count increment. This subset is (as listed above): b, c, m, n, v, x, z.

Then, after going through 10 characters, the program checks the count against 4. If the count is exactly 4, congratulations! Otherwise, try again!

And there you have it. Here is the function-obstacle-course and the final compare against 4:

0x80484a8: sub $0x62,%eax //Return from tolower and sub 0x62

0x80484ab: cmp $0x18,%eax //Compare with 0x18

0x80484ae: ja 0x80484cb //Unsigned compare -- jumps to iterate 1

0x80484b0: mov $0x1,%edx //else moves 1 into edx

0x80484b5: mov %edx,%ebx //moves edx into ebx //EDX & EBX = 1

0x80484b7: mov %eax,%ecx //EAX and EXC = 0 on first run

0x80484b9: shl %cl,%ebx //EBX = 1

0x80484bb: mov %ebx,%eax //EBX & EAX = 1

(A)0x80484bd: and $0x1501803,%eax

0x80484c2: test %eax,%eax

0x80484c4: je 0x80484cb //SO EAX MUST BE ZERO to jump

0x80484c6: addl $0x1,-0x10(%ebp) //otherwise, we ADD ONE TO A COUNT

0x80484ca: nop

0x80484cb: addl $0x1,-0xc(%ebp) //ITERATE 1 and start with next char

[…]

0x80484d5: cmpl $0x4,-0x10(%ebp) // compare count to 4!

0x80484d9: jne 0x80484f1