Alessio Mazzone
CS 0449 Wonsun Ahn
T/TH 4-5:15pm

Project 2 Part 2 write up

FILE 1:

Solution: **"tVLrZchXTGhqmGOubeXucy"**

How I got to this solution:

I first set a break point at main through gdb, and ran the program. I then disassembled the code to get the assembly code. The first thing I did with the assembly code was look for any functions that were being called. The ones I saw were: <fgets>, <chomp>,<printf>, and <puts>. My first thought was to look for some kind of strcmp function in order to see if my password would match some known password. I then typed continue, and typed a password "apple".

I could see from the assembly code that my password that I entered was stored into the $edi register after <fgets> was called. I then saw that <chomp> was called, and the results of <chomp> went somewhere. I then called nexti until I got to the instruction after <chomp> and I examined $edi with the x/s command to see if my input password "apple" had changed at all. The $edi register still held "apple", so I knew that chomp didn't alter the password I typed in at all. I still thought that there must be some comparison being done somewhere, so I continued to look at the assembly code for a compare function.

A few lines down from <chomp>, I noticed this code:

```
repz cmpsb %es:(%edi),%ds:(%esi)
seta   %dl
setb   %al
cmp    %al,%dl
```

I saw that there was a comparison being done with $al and $dl with the assembly command cmp. I used nexti to go down to cmp. I then used x/s to see if I could examine al or dl, but gdb told me that the address was out of bounds. I then looked at the line above seta and setb and saw the command repz cmpsb %es:(%edi),%ds:(%esi)was using $edi and $esi for some reason. I used x/s to check the contents of $edi and it still held "apple". I used x/s to check the contents of $esi and discovered that it held the string "tVLrZchXTGhqmGOubeXucy". I thought that this was a really random string, and that maybe the <chomp> function turned my "apple" password into "tVLrZchXTGhqmGOubeXucy" to compare later on. But, when I looked down towards the end of the assembly code, I noticed that neither $esi nor $edi were used anywhere else before the <printf> and <puts> statements. So, I thought that maybe the comparison that I had been searching for since I started analyzing this code could be the comparison between "apple" and "tVLrZchXTGhqmGOubeXucy" going on in the line
repz cmpsb %es:(%edi),%ds:(%esi).

So, I copied this strange string and reran the exacutable in gdb. I then used continue to get past the first break point in main, and I entered the string I found, "tVLrZchXTGhqmGOubeXucy". I typed next a few times, and I was greeted with a "Congratulations! Unlocked with passphrase tVLrZchXTGhqmGOubeXucy" This is how I knew that I had succeeded with the first file.

File 2:

Solution: **Any palindrome that is greater than seven characters long.**

How I got to this solution:

First I called the strings function on the alm238_2 file. I get a bunch of info, and then I see three strings:

```
PTRhp
QVhZ
[^_]
```

I ran the file with ./alm238_2 and tried all three of the strings as the passwords, but each one said "Sorry! Not correct!" So, I set a break at main, then typed the password "alessio" and that did not work. I set a breakpoint at the point where it gets my password in function d() with call 0x8048348 <fgets@plt> and it stores my password in $ebx. Then, within d(), c() is called. In c(), it compares my password to 0, so this must be a check to signal the end of the program some how. I'm guessing my password probably gets reduced to 0 characters eventually, and that this will close out the program.

I went through the method s() a few times, and I determined that it checks the length of my password and puts it into $eax. Then the function comes back to c(). Towards the end of c, the null character in my password is removed. We then move back into d(), which calls r(). Inside r(), the function checks to see if the first and last char of my password are the same. They're not, since I tried "alessio". The function then kicks me out to d(), and main and tells me my password is incorrect.

So then I tried something in which the first and last letters where the same, a palindrome, since they match the first and last letters. I typed "alessiooissela" as my password, and it told me it was correct! I then tried another palindrome, "racecar", and it said I was incorrect. That's odd, considering the other one worked. I then tried the password "aaaaaa", and it told me it was incorrect. I then tried "aaaaaaaa" and it worked. So then I tried "aaaaaaa" and it didn't work but "aaaaaaaa" did work. So this means that that the solution is a palindrome, but the palindrome has to be at least 8 characters or more.

To confirm this, I had to find somewhere that the length of the string was being compared to 7 or 8. In my function d(), four instructions above the printf statement that tells you if you are right or wrong, I found the instruction cmp $0x7,%eax. The instruction after this is jle 0x8048545 <d+93> which compares whatever is in $eax, which from above we know is our password length. It says that if the contents of $eax are less than or equal to 7, then to print out that the password is incorrect. Else, it prints that the password is incorrect. To confirm this, I set a break point at the cmp $0x7,%eax with the password "racecar", and it jumped to the end and printed incorrect. When I tried another palindrome that was longer than seven chars, "abcdedcba", it did not jump, and executed the statements after the jump, which printed I that the password was correct.

File 3:

Solutions that worked: "**4126944298**" , "**2468642111**" , "**1122468889**"

The first thing I tried with the file was running it. I ran it in gdb and typed a password, but nothing happened, and it looked like it wanted another password. So I typed in another password, and it told me they were incorrect. I tried to set a break point at main, but I guess this file does not have a function called main. So I performed an object dump of the file and piped it into a txt file. Upon examining the large volume of assembly code, I noticed that there were a high volume of getchar functions. I reran the program in gdb to see if it would take a password one character at a time. When I typed in only one character and pressed enter, it asked for another character. I repeated this, and I was able to put in 5 single characters before I was told they were incorrect. I then tried different permutations of random things. If I typed in 2 character strings like 'aa' or 'bb', it would accept 4 of them before telling me they were incorrect. If I typed in 3 character strings like 'aaa' or 'bbb', it would accept 3 of them before telling me they were incorrect. When I typed 4 characters strings like 'aaaa' or 'bbbb', it only accepted two of them before telling me they were incorrect. I tried strings longer than 4 characters, but it would only accept two of them. So, if a character is of length 4 or longer, the program only accepts two passwords.

I can see that based on the addresses in my object dump, that the object dump of section ".text" corresponds to the ??() function. While stepping though the ??() function, I was taken into another function called _dl_fixup. On line 0x00ba99c1 of this function I found a testb command that was testing the contents of $edi at an offset with 3. When I looked inside $edi with x/s , I found the letter "E". A little later, I keep seeing comparisons to $al. There seems to be a loop taking letters out of "_libc_start_main" one by one and removing them which is being kept in $edx. The loop executed once for each character is "_libc_start_main".

Looking back through the object dump, I noticed that the <getchar> was called only 10 times. So, I'll limit my guesses to 10 characters. I confirmed this when I was stepping and saw this line of code after entering some characters: `cmpl   $0xa,-0xc(%ebp)`. The comparison here is 0xa, and we know that a is 10 in hex. Right after this line, we see `jle    8048492 <tolower@plt+0x10a>`, which sends up back to get more characters if the length of the string entered is less than 10. There are comparisons being made to 0,1,8,9, and 10, which makes me think that the input should numbers. So I try a few different permutations of numbers. I typed my friend's phone number, "4126944298" and it worked.There is something about this combination of numbers that made it work. I tried a bunch of different permutations of numbers, and I also got "2468642111" and "1122468889" to work.