**Sean Clewis**

**CS 3780**

**Project 2**

**Project info:**  
For this assignment, I used visual studio. I also installed the open SSL library with vcpkg to do cryptographic hash functions and implement a pseudo random number generator.

All the tasks are embedded in a single program. The zip will include the 3 text files used to read/write accounts. As well as the source and a couple of images. The text files are prepopulated with 1000+ accounts of varying lengths. The program takes in user input to execute the corresponding commands. It is all clearly labeled in the console window when running the program. Select the corresponding number with the on-screen commands to run the desired operation. The operations include logging into an account, creating an account, generating x number of accounts in all 3 text files, and cracking the accounts (brute forcing the passwords) with a specified password length (min and max length).

The layout of the files is simple, username, salt, password. After each entry it’s simply followed by a space. In the non-salted files, there is no salt column, just username, password.

**Assignment results:**

Attempting to crack the passwords of the hashed file is relatively quick and simple for password lengths of 4 and lower. The program can almost instantly crack some passwords. However, files starting at a length of 5 become difficult and or too time consuming for my specific hardware/time constraints to brute force. Additionally, my implementation of generating the passwords is probably not the best. I generate all combinations and store them in a variable before trying to crack the files. Trying lengths of 6 and above does not work simply because it takes up all the memory on my computer and halts progress. I should probably export them to a file. That aside, since I can’t crack anything at a length of 5, it would probably be pointless anyway. Moreover, the program logs the time it takes and the number of passwords it was able to run through. For a password length of 5, the hashed file can run through 15000 passwords in a time of approximately 90 seconds. The salted version runs through 20 passwords in the same time frame with the same length. The salted version generates a salt and increments the value by 1 to the max amount allowed (which is 256) and tests a password with all salt values to attempt to crack the file. It is significantly slower than the other version.

Using a smaller password length of 3, it was feasible to crack passwords in the salted version. However, it was much slower. As it can only compare about 10 passwords every 45 seconds. Comparatively, the hashed version can run through it quickly. The salted version is more secure than just hashing the value. The hashed version brute forces approx. 375 passwords a second while the salted version yields 0.2 passwords per second. This is considering a password length of 5. The minimum password length to prevent brute forcing of this system should be about 10 characters for the salted version. The combination of a 10-character password would be 26^10/ 0.2 speed. This would roughly be 8.155 x 10^9 days. This would be far too long to try to crack it. Then considering a more realistic system with longer passwords, more characters, such as numbers and symbols, it’s easy to see the importance of making long/ secure passwords implemented with a provider who uses safe practices to store the data such as salting + hashing the password.