Emma Roskopf

I'm very sorry that this is so incomplete. I've been really sick over the weekend this was assigned and finally recovered today (the day it's due). I'm just turning in what I have done before the deadline for partial credit.

Part 1 timing: 0:08:34.958871
I was doing this on replit so I timed it by starting a count before the for loops and ending it after they were done. I couldn't find a better way to time things on replit.
I rounded up the partial millisecond because it's really close, so it took 515 seconds to run

- Number of hashes computed: 267,733 because I computed the hash then compared it to all of the user's passwords. So this is just the number of entries in the dictionary
- Time per hash: 515s/267733 hashes = 0.0019 seconds per hash
- Passwords cracked: 2805
- Time per password: 515s/2805 passwords = 0.18 seconds per password
- Passwords per number of hashes: 0.01048 passwords per hash

Part 2 timing: I ran my password2 code for around 1.5 hours then replit lost connection and didn't record anything that happened. I tried again and kept the window open and uninterrupted but it did it again. So I unfortunately have nothing to show for this part

- Number of hashes computed: nPr(267733, 2) = 71,680,691,556 hashes computed. It's a permutation because order matters, so all of the words in the dictionary pick two.
- Time per hash
- Time per password
- Etc…

Part 3 timing: I didn't have time to attempt this part

- I'm assuming though that the number of hashes computed here is the number computed before multiplied by 16^8 for the 8 digit hexadecimal number

Memory

1. Each hash string costs 32 bytes, the associated password costs 16, and it costs 32 bytes to connect to the two. So for each possible password, all 267,733 of them it costs 80 bytes to store everything and keep it connected. 267733 * 80 = 21,418,640 bytes to create this table.
2. Same process here, so 71,680,691,556 * 80 = 5734455324480 bytes to calculate the number of possible passwords, hash all of them, and index by the hash.

3. Here the fact that one password has multiple hashes makes it more complicated. So we store all of the passwords, each costing 16. Way more hashes because the salt value makes 16^8 hash possibilities and each needs to be connected to the password, so 64 per each of these salted hash strings. Assuming these are ordered somehow otherwise we need to store the salted passwords which means it's the number, 71,680,691,556, multiplied by 16^8 for the salt, multiplied by 80 for the storage of each salted password.

Give 3-4 reasons we should store password hashes and not the passwords themselves (Think in terms of threats, who the attackers might be, etc.)

1. If the file holding the password hashes is leaked, it isn't nearly as bad as leaking the passwords. No one knows the password, they just have a way to confirm the password is correct which is still not great but much better than leaking people's actual passwords. It also gives the company time to get people to change their passwords before as much damage can be done (the attackers need to figure out the passwords before they can access people's accounts, they just have a convenient way to check if their right without alerting the company that there's been a lot of attempts).
2. I did a hacking workshop one time and we used some vulnerability in the code to see what the password checker was comparing our attempted passwords to. Since for our case they wanted the password to be hackable, the plaintext password was in the code and could be extracted. By having the hashed password instead, even by looking at what the computer is comparing the attempted password to we still don't know the password. While it does give us the ability to now attempt passwords and check them on our own, it's still safer.
3. Also if we salt the password hashes, there's some protection for people who use common passwords. Their hashes are different because of the salt, so now one client's security can't be compromised by someone else using the same password. The attackers could create a table of all salt values for common passwords and find them in the password hash file, but once again that's more effort than it would be before (just Ctrl+F for "123456"). So this provides more protection regardless, even if it isn't perfect.