**CA1 C Programming Assignment Report**

Individual Contribution

For this assignment, I did the coding and debugging of task 2. This includes the study of how to create, append and implement linked lists. In my program, there are two linked lists, one for the shadow file and one for the wordlist file. The shadow file linked list consists of two char pointers, one for the user string and one for the user’s hashed password, along with a pointer to the next user. Likewise, the wordlist file linked list also consists of two char pointers, one for the password string and the hash of the corresponding password, along with a pointer to the next password.

I decided to implement linked lists for this task as I felt that linked list was the best data structure to be used, because I wanted to make my program more flexible. To elaborate, flexibility was a factor as the number of users in shadow file and the number of passwords in the wordlist file were unknown (e.g users can have more than 5 users in another shadow file), therefore I chose to implement linked lists for task 2, as elements can be inserted into linked lists indefinitely, while an array will eventually either fill up or need to be resized, an expensive operation that may not even be possible if memory is fragmented.

Secondly, I also learnt the various string library functions and implemented them into task 2. This includes strtok, strcmp and strlen.

* strtok() – used for separating a string into tokens using a delimiter, in this case the delimiter is the colon character – ‘:’, and the original string is each line of both files – shadow.txt and mytab2411.txt. The user and hash in shadow file are stored into separate variables, while the password and hash of the corresponding password is stored into two other variables using strtok. This allows it to be easily stored into the linked lists and easily referenced as well.
* strcmp() – used to compare two strings if their contents are equal to each other. This is used to compare if the hashed password of the user is equal to hash found in the wordlist file. strcmp returns an integer value, and will only return 0 if the two strings match.
* strlen() – used to find length of a string. In task 2, strlen is used to check if the hashed password length of a user is of valid length. The valid length is 26 for MD5 and 90 for SHA-512.

Lastly, I also did debugging for task 1 as we encountered issues when validating filenames and reading file line by line for the small\_wordlist.txt to hash the passwords found in it. The steps taken for debugging will be addressed later in this report.

Problems encountered

As mentioned above in task 1, we needed to validate the file name that was used as the wordlist that included the words to be hashed. At first, I use to strcmp to check if the argument of the file name is equal to the string ‘mytab2411.txt’. However, this is not flexible as the filename must be ‘mytab2411.txt’ in order to successfully run. Therefore, I use an if statement to check if the file pointer to the wordlist file is equal to NULL. This allows the user to type in any file so long if it exist in the file system allowing more flexibility.

Secondly, fgets() function was used to read the wordlist file line by line in task 1. However, the output hashes for each password was different than the expected result. The reason why the hashes were different is because fgets reads the ‘\n’ character as well. Therefore we needed to detect the ‘\n’ character of each line using isspace function and change the ‘\n’ character to the null-terminating byte – ‘\0’.

Moving on to task 2, one of the major problems I was when I was trying to put in the data (e.g user and password for shadow file) in each linked list for the two files (in my appendlinkedlist function and my appendwordlist function). At first, I put in the data as so, new\_node->usr = user, new\_node->hsh = hash. When I printed the linked list on console, it only displayed the last user, py6, on screen. I then realised that I was just setting the new\_node->usr to be equal to the pointer of the user variable, which was why it is always the user variable. I then used the function strdup which allocates new memory to hold the user string and duplicate into new\_node->usr. The linked list is late free-ed to prevent memory leak caused by strdup.

In the earlier stages of coding task 2, we also faced an issue when reading from the two files. At first, I stored each line of the files into a char array, and using a while loop to check each character of the array which equals to ‘:’. However, this was rather ineffective as I could only store the first string before the colon, for example the user of the shadow file. I found out that strtok function in the string library was much easier to separate the user and the hashed password from each line in the shadow file.

Limitations

For task 2, the limitation is that when we use a big wordlist such as the one found in /home/Misc, the program takes some time to finish running (around 5 minutes on my machine). This is because we need to append each word to the tail of the linked list from the wordlist file, thus some time is required before this can be completed. Linked lists also do not allow random access of memory; thus, we have to iterate the whole linked list containing the password and corresponding hash to see if the hash matches the user’s hash. This also takes up some time as well.