Design Document

My project works as described in the project page. My gradebook is initialized by the setup executable based off of setup_skel.c. Upon initialization, a 128-bit IV string and 128-bit key are initialized and written on the first two lines of the gradebook. When someone attempts to modify the gradebook, they need to input the correct key. Both gradebookadd and gradebookdisplay check for the correct key before any action is executed. My gradebook structure has a name field, a linked list of Students, linked list of Assignments, and a key field. The assignment structure has a field name, weight, total, and next Assignment representing the next assignment in the linked list. The Student structure had fields fname (first name), lname (last name), a linked list of Scores, and a snext Student representing the next student in the linked list. The score structure has an integer field representing the score, and then a next Score representing the next score in the linked list. When a student is added, a score of -1 is added for each assignment. This -1 is only a placeholder and is ignored when calculating a student's grade. Linked lists were used so the gradebook could easily grow without having to worry about creating new larger arrays each time or setting a max limit on the number of students or number of assignments. The information is saved and printed into a gradebook file with each piece of information separated by commas. It is similar to a csv but is not one, just a similar style. The first two lines are the IV and the key, after that there is a line for each student written as "<firstname>,<lastname>,<assignment>,<score>,<total points>,<weight>," multiple assignments are simply appended to the line in the same style. Gradebookadd and gradebookdisplay both use the read functions in data_skel.c to read in the current gradebook and create the gradebook object. This is also where I check to make sure the keys match. This information is then sent back to either gradebookadd or gradebookdisplay to perform the specified action. If it was gradebookadd and a successful change took place, this information is then sent to write_to_path in data_skel.c where the gradebook is rewritten with the updated gradebook. Neither gradebookadd nor gradebookdisplay can execute any actions without the correct key. My design check to make sure the correct key is entered before any action is executed. My design falls short in two key areas. The gradebook is not encrypted and there is not an integrity check. I will elaborate on these in the next section.

Attacks:

1. Buffer Overflow Attack from gradebook: My gradebook uses a buffer to read in lines from the gradebook. When reading in information, this is vulnerable to a buffer overflow attack. Since we are assuming an adversary can attempt to change the gradebook on the cloud, they could write the line so it overflows the buffer and possibly even get it to execute their shellcode. In order to combat this, I set my buffer very large to a size of 100000. This however, would only make it more difficult, not stop them. I make sure to only copy the number of bytes allocated for in the buffer by using strncpy instead of strcpy. This takes place in the data_skel.c file in the read_from_path method

in lines 154-186 and line 243. This prevents a buffer overflow attack from taking place when reading in the lines from the gradebook.

2. Input from attacker: Another way an attacker could try to implement a buffer overflow is through the standard input, for example, if they were to try to put overflow the buffer when reading in the name of say an assignment. In order to defend against this, I made my code use pointers for these strings and pass the pointers into other functions in order to avoid the possibility of a buffer overflow from the standard input.

3. Stealing information on the cloud: This attack was not fully implemented in my design. In this project, we are to assume that the cloud can be compromised so the data is vulnerable while on the cloud. This means, that people can access the gradebook and see the students, assignment, grades, etc. In order to prevent this, I planned to encrypt all of the data with aes-128-cbc(). I had this working correctly when encrypting information. The encrypt part is still visible in the setup_skel.c file lines 95 – 128. I double checked this with an online aes-128-cbc() encoder/decoder and so I was able to verify it was working correctly. I had to comment it out though because the decrypt was not working correctly. My idea was to decrypt when reading from the path and the code for this is still visible in data_skel.c. This was set up in the read_from_path function lines 188-216 commented out. The decrypt function I was using is above it on lines 131-148. When I would try to run this, the final step would give a "bad decrypt" error and then the wrong ciphertext would be outputted. I never was able to get it to correctly decrypt. Because I could not decrypt, I also had to comment out the encrypt part otherwise the gradebook would have been useless. Ideally, this would have worked and the information would all be written in ciphertext obtained from a secure scheme (aes-128-cbc()). This would have prevented an adversary from gaining any knowledge about the gradebook, students, or assignments without the key.

4. Integrity from attacks on the cloud: This attack was not implemented. Since we are assuming the cloud is compromised, an adversary could access the gradebook, and change the information there. Given that it was ciphertext, they wouldn't have been able to know what they were changing, but they could change it all the same. In order to provide this integrity, I was planning on using a MAC in order to provide integrity. If any information was changed on the cloud, the MAC would become invalid. The TAs and Professors would have ran this and checked the MAC each time they tried to access the gradebook. Unfortunately, I was not able to begin implementing this.