[redacted]

Project 4

Data Structures

- Gradebook:
  - A list of students (total size of 200)
  - A list of assignments (total size of 100)
  - Total number of students
  - Total number of assignments
- Assignment:
  - name of assignment
  - total amount of points
  - weight of assignment
  - a list of students who has a grade (total size of 200)
  - total number of students
- Student:
  - First name
  - Last name
  - Grade
  - Weighted grade
  - Final grade
- CmdLineResult:
  - Name of file
  - Key value
  - Action/specifier
  - Assignment/Assignment_display object
  - Student/Student_display object
- Assignment_display:
  - name of assignment
  - alphabetic flag
  - grade flag
- Student_display (gradebookdisplay version of student) :
  - first name
  - last name
- File_info:
  - IV
  - Encrypted file name
  - File name
  - Length of cipher text

Program Description

Setup.c

Within the setup program, we first check whether the file exists. If it does then we would exit. If it doesn't exist then we would create the file and then generate a key and an IV for the file. Then, we would pass in the key and IV into our encryption file to generate the encrypted version of the file. Afterwards, we would then either make or append to another file called "fileinfo" where it would store the name of file, the name of the the encrypted version of the file and the IV. To represent this, we will be making a File_info object and then appending to the fileinfo file whenever we make a new gradebook. In the end, we would then print out the key.

Gradebookadd.c

Moving onto the gradebookadd program, the idea is to first parse the inputs from the command line and store them into a struct called CmdLineResult which consists of: the filename, the key, one of the following specifiers { -AA, -DA, -AS, -DS, -AG} , an Assignment object, and a Student object.

The logic behind the parsing is to first check whether or not there is enough arguments within the command line. If there is, we then check if the order of the arguments and each of their values are valid.

- -N: check if the file with the specific filename exists or not. If it does then we store the filename into the CmdLineResult object and continue. Otherwise we exit
- -K: we iterate over each character of the specified key and check if they are in hexadecimal. If they aren't then we exit. Otherwise, we store it into CmdLineResult object and continue.

As for the specifiers, we split them up into three category/function.

Create_assingment(): In this function we will be initializing the assignment that's stored within the CmdLineResult object based on the following specifiers.

- -AA: We loop over the remaining arguments of the command line and will continue to do so if the specifiers are any of the following: {-AN, -P, -W}
    - -AN: It will check whether or not the given assignment name is alphanumeric. If it is then we will store it within the assignment object
    - -P: It will check whether or not the given points is an integer and if it is positive. If it is then we will store within the assignment object
    - -W: It will check whether or not the given weight is a float and if it is between 0 and 1. If it is then it'll be stored within the assignment object
- -DA: We loop over the remaining arguments of the command line and will continue to do so if the specifiers are any of the following: {-AN}
    - AN: It will check whether or not the given assignment name is alphanumeric. If it is then we will store it within the assignment object

Create_student(): In this function we will be initializing the student that's stored within the CmdLineResult object and will be looping over the rest of the arguments with the following specifiers {-FN, -LN}. For both of these specifiers, the given string should be only alphabetic and if they are then they will be stored within the student object's first name and last name respectively.

Add_grade(): This function is responsible to parse the command line if the specifier was "-AG". With the rest of the inputted arguments, the assignment and student object of the CmdLineResult object will be initialized and they are done so with the following specifiers: {-FN, -LN, AN, -G}

- -G: This specifier indicates to store the following argument into the grade parameter of the student object. Before storing, the given input first needs to be checked if it's a positive integer.

After finishing parsing the command line arguments, we then move on to storing the information into the file. We do this by calling the store function.

Store(): This function is responsible for adding the given information from the command line into the file. First it checks whether or not the file can be opened. Then, we will also check if the file can be decrypted with the given key within CmdLineResult. If it cannot then we will exit. If it can then we checked whether there is already an existing gradebook object within the file. If there already is a gradebook object, we read and copy it into a new gradebook object. Otherwise, initialize a new one. Afterwards, we then check which action/specifier was specified.

- -AA: We will add the stored assignment object from the CmdLineResult object to the list of assignments for the gradebook object. If it already exist then we will not add and will exit. At the end, update the number of assignments within the gradebook object.
- -DA: We will delete the specified assignment which is stored within the CmdLineResult object from the list of assignments of the gradebook object. If it does not exist within the list then we will exit. Otherwise, delete it and then adjust the list as well as decrement the number of assignments.
- -AS: We will add the specified student which is stored within the CmdLineResult object to the list of students of the gradebook object. If the student already exist then we don't add it and will exit. Otherwise, add it to the list and increment the number of students.
- -DS: We will delete the specified student which is stored within the CmdLineResult object from the list of students of the gradebook object. If the student does not exist then we exit. Otherwise, delete it from the list and adjust the list of students as well as decrement the number of students.
- -AG: We will add the specified student with a certain grade to the list of students within the specified assignment. Both should be already stored within the CmdLineResult object. If the gradebook object does not have the specified assignment within its list of assignments then we will exit. If the gradebook object does have the specified assignment but the assignment does not have the specified student within its list of students, then we will also exit. If the student already exist then we will update its grade.

At the end of the function, we will write the gradebook object back and then generate a new IV and encrypt the file.

Gradebookdisplay.c

Similarly to gradebookadd, we will also first parse the command line to get the given inputs. However, the main difference is that the specifiers for gradebookdisplay are { -PA, -PS, -PF}.

- -PA: With this specifier, we will store the assignment name and a flag that indicates where the order is alphabetic or by highest grade
  - The assignment name is first sanitized and checked whether or not it's alphanumeric
  - There will also be a check while looping through the rest of the arguments to see if there is either the specifier -A or -G
- -PS: With this specifier, we will store the specified first name and last name. Before storing, the inputs are checked whether they only contain letters.
- -PF: This specifier will indicate whether or not the print order for the final grades of all students to be by alphabetic order(-A) or by the highest grade(-G). Again, it checks whether or not there is only one of these specifiers.

Now for actually displaying the data, we first check whether or not the file can be decrypted with the given key and with the public IV of the file (from fileinfo). If it can then we decrypt and then reading the file. If the file can't be opened or is empty then we exit. Otherwise, we read and copy the gradebook object into a new one. From here, we will print the information with the specified action/specifier. If the action indicates to print out the information in a certain order (-A or -G) then we will sort the list of assignments or list of students with bubble sort.

- -PA: This means we will print out all of the students within the specified assignment. If the assignment does not exist within the list of assignments of the gradebook then we exit. Otherwise, sort the list of students for that assignment either alphabetically or by the highest grade and then print them out.
- -PS: This means we will print out all of the grades for a specific student within the gradebook. If the student does not exist within the list of students then we will exit. Otherwise go through the list of assignments and check if the student has a grade for that assignment. If the student does have a grade, then print.
- -PF: This means to calculate and print the final grade for all of the students within the student roster of the gradebook. We do this by first looping through the list of students and then for each student store and keep track of their grade for every assignment. At the end, use bubble sort to sort the students alphabetically or by their final grade.

At the end, we will generate a new IV and encrypt the file.

Attacks

1. **Stack Buffer Overflow**
   The only way for a stack overflow to happen is if we are using strcpy. Although my program is using strycpy, the input is always checked and sanitized beforehand by comparing the string. Please see line 308-383 of gradebookadd.c.
2. **Heap Buffer Overflow**
   Similar to the stack overflow, this attack would never happen within my program since we are sanitizing and checking to see if the inputs are what they should be. Because of this, whenever we store and modify the gradebook, every value is valid, otherwise it would have failed when they were being parsed from the command line. An example of this would be line 92-156 of gradebookadd.c.
3. **Plaintext Attack**
   For this attack, I was not fully able to prevent this type of attack. My program was able to generate a random key and a random IV for a specific file. Not only that but the encryption function is fully functioning. However, my decryption function is not working as properly as there are times where I'm getting segmentation faults. For now, I've commented out the use of my encryption and decryption function as it would interfere the functionality of my program. Please see line 391-398 of gradebookadd.c and line 33-64 of setup.c.
4. **Forgery**
   For this attack, my program is completely vulnerable since I have not implemented any type of authentication. The reason for this is due to the issue with my decryption algorithm where I would get errors when decrypting a file. However, if I did have the chance to implement an authentication method, I would implement a hash function that will pad a cryptographic key to create a hash-based message authentication code.