[redacted]

Project 4: Build It Writeup

System Design:

The design of my program consists of the 3 executables setup, gradebookadd and gradebookdisplay. Together these 3 executables allow legitimate users to modify/add to/view the gradebook while not allowing someone without the key to view or modify the gradebook. The setup executable creates a new gradebook file, adds an IV to the file as well as creates a random 32 byte key. This key is then outputted to stdout and also hashed and put into the gradebook file. Subsequent calls of gradebookadd and gradebookdisplay decrypt the encrypted file, modify or view the file then encrypt it and write it back out to the file. To ensure integrity, the hashed key contained in the gradebook file is compared to the hashed key that the user provided, if those hashes are not equal then the user has an invalid key. Also the decrypted plaintext is checked to see if any characters not in the character set have occurred in the gradebook, this would ensure that if someone made a change to the ciphertext (or hashed key), the program would catch that and not allow anyone else to modify the file.

The main data structure used was struct Gradebook. This struct contains two 2D array's of size 1000x100, these signify the students first and last names. I decided that at most there can be 1000 students and someone's name (first and last) can have at most 100 characters. There are also int's showing how many students and assignments are currently in the gradebook. Lastly there is an array of 100 assignments which is the second main data structure. An assignment also has 2 2D array's of size 1000x100, representing up to 1000 students who could have grades for that assignment. It also contains an int array of size 1000 that represents the students' points for that assignment. Lastly, there are variables representing the weight, total points, name and number of students who have currently taken the assignment.

The gradebook struct is the struct that is being created/updated every time gradebookadd or gradebookdisplay is called. The way it is made is by using the decrypted plaintext, which contains all the information in the gradebook, delimited by tabs and new lines. The plaintext is parsed for information and the information is added to the gradebook.

For the encryption and decryption I used AES 128 CBC mode, as found in the crypto code example. This required a 16 byte IV and 32 byte key, which were generated using the RANDOM_bytes method (not pseudorandom).

Attacks Considered:

Buffer Read Overflow:

The only time a user is allowed to read data from the file is when they call gradebookdisplay. Since this is a wrapper program and the user only can input command line arguments there isn't an easy way for the reader to read more than they are allowed. Since the user can only tell the program to print assignments, students or final grades, it is not possible for the user to access directly, and in my gradebookadd_skel.c file I make

[redacted]

sure to only allow up to 1000 students (line 326) as defined in my struct, so in my print methods I am only printing up to 1000 students, never above that. If the gradebook is full, I make sure to not allow users to add more students so the buffer will never be over read or over written.

Buffer Write Overflow:

This attack could happen if a user tried to get the program to write past where it was supposed to in memory. Since the user is only allowed to input command line arguments, we can check the length of all those arguments and make sure that the user didn't input a first name, last name, assignment name, etc. that's too long. The way I did this was by using the strlen function before I put any string passed in from the user into a buffer. If the string was determined to be too long (i.e. a first name of length 500 or gradebook name of length 101) then the program doesn't accept this inputs and tells the user it's invalid. I checked the length of user input many times in the project, a couple examples are in gradebookadd_skel.c lines 64, 80, 109.

Code Injection:

A user could try and perform a code injection using the command line arguments, but since all strings have a strict length there is no way for the user to write passed that length, as described in the section above. I make sure of this anytime the user inputs strings for their command line arguments, checking the strlen and making sure it's less than 100 characters (I am assuming names are <100). I check this in gradebookadd_skel.c lines 64, 80, 109.

Integer Overflow:

The user is allowed to input some integers as well as floating points in normal use. Midterms have points and weights associated with them and students have grades. The way I prevented against this attack is to validate user input so that integers are at most 9999 (maximum points for an assignment, max points for a student on an assignment) and the floating points for weight should have at most 5 characters (0.000-0.999). I do these checks multiple times, for example in gradebookadd_skel.c lines 123, 137.

Format String:

I do not allow users to ever enter the percentage sign as no valid input should contain this character (names should not contain %), I check this in data_skel.c line 252. I made a charset of all valid characters that could be in the file, this includes a-z, A-Z, 0-9, parenthesis, dashes, periods, commas, spaces, new lines and tabs. If we see anything outside of this either a valid user has inputted something invalid or an attacker has changed the ciphertext.

Man in the Middle:

[redacted]

If an attacker wants to try and gain information about the gradebook, they will only be able to see the IV, size of the ciphertext, encrypted ciphertext and hashed key. The attacker may change characters in the ciphertext but my code checks for this. The way I check for this is by making sure that the decrypted plaintext only contains allowed characters (mentioned above line 252 data_skel.c). If an attacker did change the ciphertext (or changes the hashed key), when the text is decrypted there is a large chance that the decrypted plaintext will contain a non-normal character since CBC mode uses block chaining so everything after the change would be corrupted. I was not able to completely prevent this attack, as if a user changed the IV there is no way for the program to know because sometimes if the IV is slightly different the program will still encrypt/decrypt normally. Also, if the user changes the size of the ciphertext this could break the program although I wasn't able to make this happen myself using a hex editor. If I had more time I would have done would be to use a MAC on the whole gradebook file, appending the correct code to the file as well. Then the user would run the MAC on the file and compare it to the correct code to see if anything changed.