Project 4: Build-it Secure Gradebook

[redacted]

The project consists of 3 programs: setup, gradebookadd, and gradebookdisplay. The programs share common libraries, namely, crypto.h and data.h.

Setup

Example usage: setup -N gradebookname

The setup program initializes a new gradebook with the name “gradebookname”. If the gradebook already exists, it returns an error. The initial gradebook consists of the text “numStudents 0 numAssignments 0 “. The program generates a cryptographically secure random IV and key with the openssl functions RAND_bytes and encrypts the gradebook using aes_128_cbc. The program then calculates a MAC of the iv and ciphertext to provide authenticity and integrity. The MAC, IV, and ciphertext are then stored in the file with name “gradebookname” and the secure key is output to the console.

Gradebookadd

The gradebookadd program must be called with specific options from the set {-N, -K, -AA, -DA, -AS, -DS, -AG, -AN, -FN, -LN, -P, -W, -G}. I refer to the action set as {-AA, -DA, -AS, -DS, -AG}. The action option set is {-AN, -FN, -LN, -P, -W, -G}. Each action option for gradebookadd requires an additional action option specifier.

The usage is: gradebookadd -N gradebookname -K 0e12bc83 {action} {action options/ specifiers}.

When the program is run, it calls the function parse_cmdline with an addType flag set. The function parses the program argument variables and checks whether the correct order and format have been specified. The addType flag allows the program to search for a valid action from the set {-AA, -DA, -AS, -DS, -AG}. Once a valid action has been found, the parse_cmdline function calls the parse_action_option function, where it searches for action options in the action option set, then validates the action options found based on the action. The specified gradebookname, key, action, action options, and action option specifiers are stored in a CmdLineResult structure.

Once the program arguments have been parsed, the program calls the gradebook_authenticate function, which it checks whether the gradebookname file exists, and then reads the file and computes the MAC of the given IV and ciphertext with the input key. It then compares the computed MAC with the stored MAC in the file and if they match, it returns a valid authentication.

After the gradebook has been authenticated, the program calls the decrypt_and_read_gradebook function. For this, it decrypts the gradebookname file with the input key and generates a Gradebook structure. The gradebook structure contains the number of assignments, number of students, a pointer to a linked list of assignments, and a pointer to a linked list of students. The program allocates memory for the linked list based on the number of assignments and students that where specified in the gradebook and then creates the list and fills in the data in the list. An assignment node consists of the assignment name, number of points, and weight. A student node consists of the student first name, last name, a dynamic array of grades, and a final grade. The index of an assignment grade in the grade array corresponds to the location of the assignment in the assignment linked list.

After the gradebook structure has been made, the program calls the modify_gradebook function. Here, the function uses the parsed command line result to decide what setting of gradebook modification to use. The settings either add an assignment, delete an assignment, add a student, delete a student, or add a grade. Based on the type of action, the program adds or deletes a node in the assignment or student linked list or updates the grade array for a specific student.

Once the gradebook structure has been modified, the program calls the encrypt_and_write_gradebook function. Here, a dynamic character array is created and formatted for the gradebook. An example of the format of the gradebook is shown below:

```
numStudents 2 numAssignments 3
Test 50 0.3
Project 25 0.4
HW1 10 0.2
John Smith 49 24 9
Russel Tyler 40 19 10
```

The encrypt_and_write_gradebook function prints the gradebook structure data in the format specified above in a dynamic character array. The character array is then encrypted using the input key and with a new IV. Once the gradebook is encrypted, a MAC in computed over the ciphertext and IV. The MAC, IV, and ciphertext are then printed to the specified gradebook file.

GradebookDisplay

The gradebookdisplay program must be called with specific options from the set {-N, -K, -PA, -PS, -PF, -AN, -FN, -LN, -A, -G}. I refer to the action set as {-PA, -PS, -PF}. The action option set is

{-AN, -FN, -LN, -A, -G}. The action options {-AN, -FN, -LN} require additional action option specifiers, while the action options {-A, -G} do not.

The usage is: gradebookdisplay -N gradebookname -K 0e12bc83 {action} {action options/ specifiers}.

The gradebookdisplay program runs similarly to the gradebookadd program at first. It calls the parse_cmdline function with a displayType flag. With the displayType flag set, the program searches for valid actions in the set {-PA, -PS, -PF}, then searches for valid action options in the action option set. If an action option in the set {-AN, -FN, -LN} is found, it checks for an action option specifier. If an action option in the set {-A, -G} is found, it continues to look for the next action option. Once, the action options and specifiers have been found, the program checks if the action options are valid for the specified action. The gradebook name, key, action, action options, and action option specifiers are stored in a CmdLineResult structure.

After the command line has been parsed, the program calls the gradebook_authenticate function. This function reads in the MAC, IV, and ciphertext of the gradebook file and computes the MAC of the IV and ciphertext with the input key. It then compares the computed MAC with the given MAC to authenticate and validate the gradebook file.

If the gradebook is authenticated and the computed MAC matches with the MAC in the gradebook file, the program calls the decrypt_and_read_gradebook function. This function is exactly the same as the function in the gradebookadd program. It decrypts the gradebook file and stores the gradebook information in a gradebook structure that contains the number of students, number of assignments, a pointer to a linked list of assignments, and a pointer to a linked list of students.

After the gradebook is decrypted and read, the program calls the print_gradebook function. This function decides what action to perform based on the action stored in the CmdLineResult structure. The actions it performs are either print assignment, print student, or print final. Then depending on the {-G, -A} action option, the gradebook is printed to the console in alphabetic or grade order. If the -G action option is specified, the student linked list is sorted based on a specific assignment grade from highest to lowest. If the -A action option is specified, the student linked list is sorted based on alphabetic order of the student last names and first names. If the -PF action is used, the program goes through the student linked list and calculates the student final grade based on the student grade array and the weights and points of each assignment. The final grade of each student is stored in each student node in the linked list. If the -G action option is specified when the -PF action is used, the students are sorted based on the final grade calculated. After the linked lists have been sorted, the gradebook is printed to console based on the specified action and action options.

Attack Defense

Attacks that I have considered are buffer overflow attacks, existential unforgeability under chosen message attacks, chosen plaintext attacks, format string vulnerability attacks, and stale memory/ dangling pointer attacks.

For buffer overflow attacks I refer to the file data_skel.c in the check_valid_action_option_specifier function line 244. Here I check whether the argument variable has a string length greater than the maxLength defined. If the string length is greater than the maxLength, then I memcpy the argument string with length maxLength. If the string length is less than the maxLength, then I strcpy the argument string. Doing this prevents the string from being too large to prevent stack smashing attacks. Another place I consider buffer overflow attacks is on line 626 in the data_skel.c file. For this I have declared a const array of strings of the valid action options. If a valid action option input is found, instead of using the user input action option, I use the string declared in the const defined string array. This way, I am not actually storing the user input actions in the CmdLineResult structure.

For preventing forgery attacks, I have used a MAC on the IV and ciphertext. For this, I refer to line 383 in the gradebookadd_skel.c file, line 293 in the gradebookdisplay_skel.c file, and line 16 in the crypto.c file. Here, I compute the MAC over the entire IV and ciphertext. The input key of the user is used and the output MAC is compared with the MAC stored in the gradebook file. If the MAC's match, then we can verify that the gradebook has not been maliciously modified by unwanted parties that do not have the security key.

For preventing chosen plain text attacks I refer to line 58 in the crypto.c file and line 574 in the gradebookadd_skel.c file. We use an aes 12 bit CBC encryption to prevent attackers from finding out information about the student grades. A cryptographically secure key and IV are generated with the openssl RAND_BYTES function and the key is used to encrypt the file. With this, as long as the malicious parties do not have the key, then the gradebook information cannot be accessed.

Format string vulnerabilities are prevented due to the checking of proper inputs on the command line arguments. Here, I refer to the function at line 71 in the data_skel.c file. Here, I check the input options of the user input and return invalid if the user input is not a valid character. This prevents an attacker from inputting format strings and causing buffer overreads, etc.

For preventing dangling pointer and stale memory attacks, I have freed memory and set the pointers to NULL when necessary. I refer to line 252 in the gradebookadd_skel.c file. When deleting a student, I make sure to set the freed student node to NULL to prevent attackers from reallocating the memory under their control. I also refer to line 135 in the same file when deleting an assignment.

Other attacks I have considered are chosen ciphertext attacks, which are prevented because of the encrypt then authenticate scheme that was implemented. Overall, an attack should not be able to query the gradebook with either of the programs to find out student and assignment information. The programs also should not accept bogus gradebook files because of the MAC scheme that was implemented. Without a correct key, the attacker should not be able to access or modify the gradebook.