Secure Gradebook

Overview

The secure gradebook file system is a wrapper program with three components: *setup*, *gradebookadd*, and *gradebookdisplay*. When initializing a new gradebook, a single key is generated. An authenticated user may modify the gradebook by adding/deleting assignments, adding/deleting students, or adding grades. The user can also display the gradebook's contents by specifying a print option.

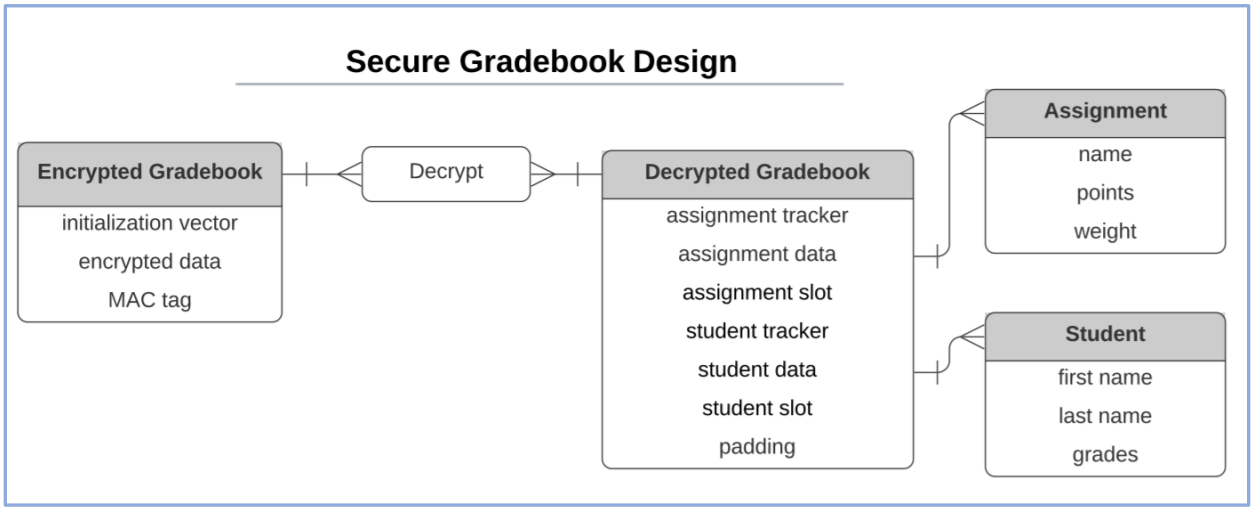
System Design

In the following sections, the overall design of the gradebook file system is described.

Gradebook File Format

The gradebook is structured as shown in the figure below. When a gradebook is initialized, a decrypted gradebook is set to all zeros (i.e., {0}). Each gradebook is set to accept a maximum of 20 assignments and 50 students, as defined in *data.h.* Thus, the file is instantiated to a particular size and prevents an overflow of assignments and students via an assignment tracker and a student tracker. The gradebook contains an array of assignments of type Assignment and an array of students of type Student. Each Assignment contains the assignment name, which can have a maximum length of 100 characters, the assignment points, and the assignment weight. Each Student contains a person's first and last name, each allowing a maximum length of 100 characters, and an array of grades of maximum size 20.

A gradebook is always saved in its encrypted format within the file system. It contains an initialization vector, the ciphertext gradebook, and a MAC tag. The encrypted data gets decrypted when a user wants to modify or display the gradebook's contents, and it is formatted in the same order as the Decrypted Gradebook block below.



Programs

Setup

The *setup* program initializes a new encrypted gradebook with a specified name and will generate a single cryptographically strong pseudorandom key that allows access to the gradebook for authorized parties. When a new gradebook is added to the file system, it is encrypted using AES 128-bit CBC encryption and then authenticated using a keyed-hash MAC (i.e., HMAC-MD5). The same key is used to encrypt and to key the hash function.

Gradebookadd

The *gradebookadd* program first parses the command line arguments and sanitizes the input by blacklisting unwanted entities (i.e., invalid argument count and invalid string lengths). Afterwards, the input is validated by whitelisting a list of trusted entities, such as the gradebook file name and the set of allowed options (i.e., {-N, -K, -AA, -DA, -AS, -DS, -AG, -AN, -FN, -LN, -P, -W, -G}). If all user input passes the verification stage, then the gradebook file gets read in and the input key and MAC tag obtained from the encrypted gradebook are used to authenticate the user. The tag from the end of the encrypted gradebook file is compared with the generated HMAC tag on the encrypted data, and if the contents are the same, the data is then decrypted using the same input key. If decryption succeeds, the gradebook gets modified. Before storing the updated gradebook back into the file system, it goes through the encryption-then-authentication scheme again.

Gradebookdisplay

The *gradebookdisplay* program first parses the command line arguments and sanitizes the input by blacklisting unwanted entities. Afterwards, the input is validated by whitelisting a list of trusted entities, such as the gradebook file name and the set of allowed options (i.e., {-N, -K, -PA, -PS, -PF, -AN, -FN, -LN, -A, -G}). If all user input passes the verification stage, then the gradebook file gets read in and the input key and MAC tag obtained from the encrypted gradebook are used to authenticate the user. The tag from the end of the encrypted gradebook file is compared with the generated HMAC tag on the encrypted data, and if the contents are the same, the data is then decrypted using the same input key. If decryption succeeds, the specified gradebook contents are displayed to the user.

System Security

This section describes four attacks that were considered when designing and implementing the secure gradebook file system and the measures that the implementation takes to defend against those attacks.

Injection Attacks

Injection attacks consist of injecting code that is then interpreted and executed by the application. This type of attack exploits poor handling of untrusted data and are usually made possible due to a lack of proper input data validation (i.e., allowed characters, data format, and amount of expected data). To defend against bad input, the process of testing input received is implemented. Examples of input sanitization can be observed in the parse_cmdline() function (lines 46-459) in *gradebookadd.c* and the parse_commands() function (lines 33-281) in *gradebookdisplay.c*.

Buffer Overflow Attacks

In the case of languages, such as C, reading from and writing to memory allocations does not entail any automatic bounds checking. In other words, there is no check that the number of bytes to be written or read will actually fit into the buffer in question. Thus, the program can overflow the capacity of the buffer. This results in data being written past its end and overwriting the contents of subsequent addresses on the stack or heap. To mitigate this attack, I looked for areas where unsafe behavior of not checking any bounds on the target buffer could lead to exploitation. Using C functions associated with the strn- version was a safer option than the str- version as only strings of the maximum size could be written to the target buffer. Examples of using these functions are displayed in each one of the programs (*note: use Ctrl+F to find instances of* strncpy *and* strncmp).

Format String Attacks

In *gradebookdisplay.c*, measures are taken to prevent an attacker from executing a format string vulnerability attack on the system. Every time a string is printed to stdout, a format string is always passed to printf to correctly display the gradebook's contents. Examples of this are found on lines 354, 409, 437, 510, and 551 of *gradebookdisplay.c*. This mitigates format string attacks because it is possible that a malicious user provides a format string as a person's first name or last name when using *gradebookdisplay*. In this case, if "%s" was not used (i.e., printf(gradebook->students[i].lastname)), then the string gradebook->students[i].lastname could potentially reveal undesired information about the gradebook file to an attacker.

Confidentiality Attacks

The purpose of confidentiality is to ensure the protection of data by preventing unauthorized disclosure of information. Only users with legitimate authorization to access the information should be permitted to modify or display it. However, a malicious user might attempt to crack the encryption and decryption schemes. To prevent an attacker from learning facts about the names of students, assignments, or grades of individuals by inspecting the gradebook file itself, the gradebook system uses an AES-described symmetric-key algorithm to encrypt and decrypt the file. To encrypt the plaintext, I generated a cryptographically strong pseudorandom initialization vector (IV) using OpenSSL's RAND_bytes() function every time (as can be seen on line 800 in *gradebookadd.c*). The IV is appended to the beginning of the file to be used in the decryption process. This process guarantees a different ciphertext every time an encryption occurs and therefore reinforces confidentiality by preventing an attacker from assuming anything about the plaintext.

Integrity Attacks

Since users of the secure gradebook file system have access to the gradebook file itself, a malicious user could attempt to modify the gradebook file without using *gradebookadd*. If a malicious user was able to do this, then they could modify the data, causing *gradebookdisplay* to output incorrect information. To defend against this attack, the gradebook system uses an HMAC to authorize gradebook access. HMAC has strong properties of being unforgeable under chosen-message attacks and being a pseudo-random function, so it strongly protects against an attacker, who does not have the key, from learning information about the contents of the encrypted gradebook. The encrypted gradebook includes a tag along with the encrypted gradebook data and IV. This is initially done after the encryption process of the *setup* program. After encrypting the plaintext using CBC mode, I use the OpenSSL EVP library to generate a tag (line 108 of *setup.c*) and write it to the end of the encrypted gradebook file (line 111 of *setup.c*). This tag is used for verification before the decryption process of the *gradebookadd* and *gradebookdisplay* programs (lines 769-794 of *gradebookadd.c* and lines 611-636 of *gradebookdisplay.c*). A comparison of the tags from the end of the encrypted gradebook file and the tag obtained from HMAC on the encrypted data determines if integrity has been violated.