Project 4A: Build-It

Due November 20, 5:00 PM

Note: This is a **group project**; you can work in groups of at most 4. This is part one; you have two weeks to work on the Build-It portion, and then will have one week to work on the Break-It portion.

1 Overview

A professor and two TAs all want to be able to modify a gradebook, so they want to store it on a public cloud as opposed to locally. However they do not trust the security of the cloud. Therefore, each time they want to modify the gradebook, they will download it, modify it on their local computer and re-upload it (you may assume that they always download/modify/upload the gradebook one at a time). They will create a wrapper program that is run on their local computers. The wrapper program will have three components:

1. `setup`, which will initialize a new gradebook with a specified name and will generate a single key that allows access to the gradebook for authorized parties. Copies of the same key will be stored locally on the computers of the Prof and TAs.

2. `gradebookadd`, which will allow adding a new student or assignment, or entering a new grade for a student and assignment already contained in the gradebook.

3. `gradebookdisplay`, which provides a few ways in which students' grades can be printed out.

Before executing an add or display query, both `gradebookadd` and `gradebookdisplay` will check whether the gradebook has been modified by an unauthorized party who does not hold the key that is associated with the gradebook. If such "tampering" is detected, an error message will be printed out.

Your goal is to write a secure wrapper program that prevents an attacker, who does not have the key, from learning information about the contents of the gradebook (confidentiality) or from modifying the gradebook without being detected (integrity). The threat model assumes that the cloud is fully compromised, allowing an attacker to read or modify any files stored there.

Your program will be evaluated based on *correctness* tests we run on it, the design document you submit justifying your design choices, and whether other students successfully attack your code during the Break-It phase (In the Break-It phase, you will receive points for a successful attack on other students' implementations, **but will not lose points for other students' attacks on your implementation**. Extra credit may be assigned (based on the Instructors' discretion) for submissions that perform well during the Break-It phase). During Break-It, a successful attack launched in the above threat model (where only the cloud is compromised) will automatically receive points. Attacks that require access to the local computers of the Prof and/or TAs will be considered on a case-by-case basis. In particular, students need to explain why their attack is made possible by a vulnerability in the gradebook wrapper program itself (as opposed to an attack that is always possible, even when there is no vulnerability in the wrapper program, such as stealing the key). E.g., if a malformed input to the wrapper program causes a buffer overflow that can be used to perform code injection, this would be a valid exploit.

You will build the most secure implementation you can; then you will have the opportunity to attack other students' implementations.

2 Getting Set Up

This homework assignment uses Docker. Once you have Docker installed, you can use the **preconfigured Docker image we have given you**, available here. See the HW1 spec if you need a refresher for using Docker.

You must write your implementation in either C or Java. We are providing basic starter code for both, including Makefiles. You can download the starter code here.

Sample input/output sequences are provided here. These sequences are tested by the five public tests for this project.

The starter code includes a test script `test_script_template.sh` that tests the five public tests for this project. Running the script should generate a file `YOUR_FOLDER_NAME.log` inside the directory of the project. You can open that and see the output of your program vs. the expected output (from the sample input/output we provided).

We encourage you to use any external libraries that are helpful (crypto libraries included). **Remember: do not write your own crypto.**

3 Programs

You will design a gradebook format and implement `setup` as well as both `gradebookadd` and `gradebookdisplay` to use it.

- The `setup` program generates a key and an empty gradebook with a specified name. The program returns the key to the user.
- The `gradebookadd` program receives as input the name of a gradebook and a key. The program adds data to a gradebook.
- The `gradebookdisplay` program receives as input the name of a gradebook and a key. The program displays data from the log.

3.1 `setup`

This is a command line program that is executed by running `setup`. It initializes a new gradebook with the specified name, generates and outputs a cryptographic key to stdout.

There is only a single allowed option -N that specifies the name of the generated gradebook.

- `-N` The name of the gradebook file. The filename may be specified with a string of alphanumeric characters (including underscores and periods). The gradebook file will be generated in the same directory in which `setup` is run. If a file of this name already exists, an error occurs. Specifically, `setup` should print "invalid", and exit with error code 255.

Example Usage:

- `setup -N mygradebook`

3.2 `gradebookadd`

This is a command line program that is executed by running `gradebookadd` with some subset of specified options and corresponding values.

The set of allowed options is `{-N, -K, -AA, -DA, -AS, -DS, -AG, -AN, -FN, -LN, -P, -W, -G}`. Different sets of options and corresponding values may or may not be valid. If a set of options/values is invalid, `gradebookadd` should print "invalid," and exit with error code 255. Otherwise it will add the data to the gradebook.

We next explain the various options and their purpose:

- A file name must be specified using the following option:
  - `-N` The gradebook's filename may be specified with a string of alphanumeric characters (including underscores and periods). The filename must be located in the same directory that the `gradeboodadd` program is running in. If the file does not exist, it causes an error.
- A key must be specified using the following option:
  - `-K` The key may be specified with a string of hex digits.
- An action must be specified using the following options:
  - `-AA` *add-assignment* Add a new assignment.
  - `-DA` *delete-assignment* Delete an assignment for all students.
  - `-AS` *add-student* Add a new student.
  - `-DS` *delete-student* Delete a student.
  - `-AG` *add-grade* Add a new grade for an existing student and assignment.
- If the action is *add-assignment* then the following additional options must be specified:
  - `-AN <assignment-name>` Name of assignment. Names are alphabetic characters (a-z, A-Z) in upper and lower case and numbers (0-9). Names may not contain spaces. Names are case sensitive.
  - `-P <assignment-points>` Number of points assignment is out of (required to be a non-negative integer)
  - `W <assignment-weight>` Weight of assignment when computing total grade (required to be a real number in [0,1])
  - If an assignment with the same name already exists then an error occurs.
  - If the sum of the weights of the new assignment and all the current assignments adds up to more than 1 then an error occurs.
- If the action is *delete-assignment* then the following additional options must be specified:
  - `-AN <assignment-name>` Name of assignment. Names are alphabetic characters (a-z, A-Z) in upper and lower case and numbers (0-9). Names may not contain spaces. Names are case sensitive.
  - If an assignment with the specified name does not exist then an error occurs.
- If the action is *add-student* then the following additional options must be specified:

  - `-FN <student-first-name>` First name of student. Names are alphabetic characters (a-z, A-Z) in upper and lower case. Names may not contain spaces. Names are case sensitive.
  - `-LN <student-last-name>` Last name of student. Names are alphabetic characters (a-z, A-Z) in upper and lower case. Names may not contain spaces. Names are case sensitive.
  - If a student with the same first and last name already exists then an error occurs.

- If the action is *delete-student* then the following additional options must be specified:
  - `-FN <student-first-name>` First name of student. Names are alphabetic characters (a-z, A-Z) in upper and lower case. Names may not contain spaces. Names are case sensitive.
  - `-LN <student-last-name>` Last name of student. Names are alphabetic characters (a-z, A-Z) in upper and lower case. Names may not contain spaces. Names are case sensitive.
  - If a student with the specified first and last name does not exist then an error occurs.

- If the action is *add-grade* then the following additional options must be specified:
  - `-FN <student-first-name>` First name of student. Names are alphabetic characters (a-z, A-Z) in upper and lower case. Names may not contain spaces. Names are case sensitive.
  - `-LN <student-last-name>` Last name of student. Names are alphabetic characters (a-z, A-Z) in upper and lower case. Names may not contain spaces. Names are case sensitive.
  - `-AN <assignment-name>` Name of assignment. Names are alphabetic characters (a-z, A-Z) in upper and lower case and numbers (0-9). Names may not contain spaces. Names are case sensitive.
  - `-G <grade>` Number of points student received (required to be a non-negative integer).
  - If a student with the provided first name/last name does not exist or an assignment with the provided name does not exist then an error occurs.
  - If a grade for this student and this assignment was previously specified, then the newly supplied grade should replace the previously specified grade. This should *not* cause an error.

Command line arguments must appear in the following order:

1. A gradebook name must be specified first with `-N`
2. A key must be specified second with `-K`
3. An action must be specified third with exactly one of `{-AA, -DA, -AS, -DS, -AG}`
4. After the action option, other arguments from the set `{-AN, -FN, -LN, -P, -W, -G}` may be specified in any order and if the same argument is provided multiple times, the last value is accepted.

An incorrect order of arguments results in an error.

After `gradebookadd` exits, the gradebook specified by the `-N` argument should be updated. The added information should be accessible to the `gradebookdisplay` tool when the key provided to both programs is the same, and not available (e.g., by inspecting the file directly) otherwise.

3.2.1 Return values and error conditions

If `gradebookadd` must exit due to an error condition, or if the argument combination is incomplete or contradictory, the program should print "invalid" to stdout and exit, returning a 255.

If the gradebook to be appended already exists, and the supplied key does not match the expected key, "invalid" should be printed to stdout and 255 returned. (When a gradebook file is first created, the generated key becomes the valid key for all future accesses to that gradebook.)

Some examples of potentially valid orderings and combinations of command-line options:

- `gradebookadd -N mygradebook -K 45ea448f -AA -AN midterm -P 100 -W 0.25`
- `gradebookadd -N mygradebook -K 45ea448f -AS -FN John -LN Smith`
- `gradebookadd -N mygradebook -K 45ea448f -AG -AN midterm -FN John -LN Smith -G 95`

Some examples of conditions that would result in printing "invalid" and doing nothing to the gradebook:

- The specified student or assignment does not exist when executing an "add grade" action.
- The name for a student or assignment does not correspond to the character constraints
- Conflicting command line arguments are given, for example both `-AA` and `-AS`
- An invalid combination of command line arguments are given, for example -AA with -FN or a command line argument is missing, for example `-AA` is specified but no `-AN` option is provided.

3.3 `gradebookdisplay`

This is a command line program that is executed by running `gradebookdisplay` with some subset of specified options and corresponding values.

The set of allowed options are `{-N, -K, -PA, -PS, -PF, -AN, -FN, -LN, -A, -G}`. Different sets of options and corresponding values may or may not be valid. If a set of options/values is invalid, `gradebookdisplay` should print "invalid," and exit with error code 255. Otherwise it will print out the requested data.

We next explain the various options and their purpose:

- A file name must be specified using the following option:
  - `-N` The gradebook's filename may be specified with a string of alphanumeric characters (including underscores and periods). The filename must be located in the same directory that the `gradebookadd` program is running in. If the file does not exist, it causes an error.

- A key must be specified using the following option:
  - `-K` The key may be specified with a string of hex digits.
- An action must be specified using the following options:
  - `-PA` *print-assignment* Prints out grades of all students for a particular assignment. The print out should consist of tuples (Last Name, First Name, Grade), with one tuple per line. There will be two options that allow printing grades either alphabetically by last name, first name, or by grade from highest to lowest.
  - `-PS` *print-student* Prints out all grades for a particular student. The print out should consist of pairs (Assignment Name, Grade), with one tuple per line.
  - `-PF` *print-final* Prints out final grades for all students. These are computed by multiplying the weight of each assignment by the grade and taking the sum. The print out should consist of tuples (Last Name, First Name, Grade), with one tuple per line. There will be two options that allow printing grades either alphabetically by last name, first name, or by grade from highest to lowest.
- If the action is *print-assignment* then the following additional options must be specified:
  - `-AN <assignment-name>` Name of assignment. Names are alphabetic characters (a-z, A-Z) in upper and lower case and numbers (0-9). Names may not contain spaces. Names are case sensitive.
  - `-A` Alphabetical order. Tuples are printed in alphabetical order, ordered by Last Name of student and then First Name of student (assuming there are repeat last names).
  - `-G` Grade order. Tuples are printed in grade order from highest to lowest.
  - If an assignment with the specified name does not exist then an error occurs.
  - Exactly one of `-A, -G` should be specified. Otherwise an error occurs.
- If the action is *print-student* then the following additional options must be specified:
  - `-FN <student-first-name>` First name of student. Names are alphabetic characters (a-z, A-Z) in upper and lower case. Names may not contain spaces. Names are case sensitive.
  - `-LN <student-last-name>` Last name of student. Names are alphabetic characters (a-z, A-Z) in upper and lower case. Names may not contain spaces. Names are case sensitive.
  - If a student with the specified first and last name does not exist then an error occurs.
- If the action is *print-final* then the following additional options must be specified:
  - `-A` Alphabetical order. Tuples are printed in alphabetical order, ordered by Last Name of student and then First Name of student (assuming there are repeat last names).
  - `-G` Grade order. Tuples are printed in grade order from highest to lowest.
  - Exactly one of `-A, -G` should be specified. Otherwise an error occurs.

Command line arguments may appear in the following order:

1. A gradebook name must be specified first with `-N`

2. A key must be specified second with `-K`
3. An action must be specified third with exactly one of `{-PA, -PS, -PF}`
4. After the action option, other arguments from the set `{-AN, -FN, -LN, -A, -G}` may be specified in any order and if the same argument is provided multiple times, the last value is accepted. An incorrect order of arguments results in an error.

3.3.1 Return values and error conditions

If gradebookdisplay must exit due to an error condition, or if the argument combination is incomplete or contradictory, the program should print "invalid" to stdout and exit, returning a 255.

If the gradebook already exists, and the supplied key does not match the expected key, "invalid" should be printed to stdout and 255 returned. (When a gradebook file is first created, the generated key becomes the valid key for all future accesses to that gradebook.)

Some examples of potentially valid orderings and combinations of command-line options:

- `gradebookdisplay -N mygradebook -K 45ea448f -PA -A -AN midterm`
- `gradebookdisplay -N mygradebook -K 45ea448f -PS -FN John -LN Smith`
- `gradebookdisplay -N mygradebook -K 45ea448f -PF -G`

Some examples of conditions that would result in printing "invalid":

- The specified student or assignment does not exist when executing a print student or print assignment action.
- Conflicting command line arguments are given, for example both `-PA` and `-PS`
- An invalid combination of command line arguments are given (e.g., `-PF` with `-FN`) or a command line argument is missing (e.g., `-PA` is specified but no `-AN` option is provided).

3.4 Other Restrictions

Your programs are **not permitted** to write extra files to disk.

4 Security Model

The system, as a whole, must guarantee the confidentiality and integrity of the gradebook in the presence of an adversary that **does not know the key** (i.e. an adversary who corrupts the server, but not the local computers of the Prof and TAs). Recall that the key is used by both the `gradebookadd` and `gradebookdisplay` tools, and is specified on the command line. *Without knowledge of the key* an attacker should *not* be able to:

- Query the gradebook via `gradebookdisplay` or otherwise learn facts about the names of students, assignments, or grades of individuals by inspecting the gradebook file itself.
- Modify the gradebook via `gradebookadd`.

- Fool `gradebookdisplay` or `gradebookadd` into accepting a bogus gradebook file. In particular, modifications made to the gradebook by means other than correct use of `gradebookadd` should be detected by (subsequent calls to) `gradebookadd` or `gradebookdisplay` when the correct key is supplied.

5 Team Requirements

Form teams of **up to 4 members**. You can have teams with fewer members as well, but not more than 4 members.

After forming your group, complete this Google Form. Please make sure you include **team name, participant names, and emails**. This is required for making sure that everyone has access to the GitLab repo to submit your code. TAs will create a private repo for your team on Gitlab.

6 Deliverables

The Build-It part of this project will be graded in two parts:

- **Correctness Tests (50 points)**: There are 15 automated correctness tests that your submission is tested upon: 5 public and 10 secret. We are providing the script we will use to test the 5 public tests.
- **Design Document (150 points)**: A design document should be submitted as a PDF document in the team GitLab repository. It should address the following key points:
  - Describe your overall system design in sufficient detail for a reader to understand your approach without reading the source code directly. This must include a description of the format of your gradebook.
  - List **four** specific attacks you have considered, and describe how your implementation counters these threats. Please note the relevant lines of code for your defense. If you were not able to completely (or at all) prevent a threat you identified, you may still mention it. In that case, describe any partial mitigation you implemented and explain anything you wanted to implement but were for whatever reason unable to. Please be clear about distinguishing what you have implemented vs. what you would have implemented. This is critical for how we will be grading this part of the project (see Grading below).
  - List the specific contributions of each team member.

Submission will be done through the GitLab repositories we provide to teams. All code and your design document must be pushed to your repo by the due date. Place your code (including Makefile) and design document in a directory called `build` in your repository. Your submission should be set up so that the `setup`, `gradebookadd`, and `gradebookdisplay` programs are all created when we run `make` in your build directory.

If you are using Java with external libraries, you will likely need to modify the sample makefile. Use the `-classpath` option for both `javac` and `java`.

Note that due to the nature of Build-It/Break-It, **we are not allowing any late days for this project**.

7 Grading

Build-It will be worth 200 points:

- 50 points for the automated correctness tests.
- 150 points for your design document:
  - 20 points for your description of your approach.
  - 30 points for each of the four specific vulnerabilities you defend against. For maximum points, your defense should be correct, fully implemented, and well explained. Well explained attacks that are not fully implemented will receive partial credit.
  - 10 points for the list of team member contributions.
  - Extra credit will be permitted for one extra attack/countermeasure beyond the original four.