CMPSC 447 - Fall 2022 - Project 1: 2FA Questions

1 Dates

- **Out:** *January 18, 2022*
- **Extra Credit Deadline:** *January 31, 2022, 11:59pm*
- **Due:** *February 5, 2022, 11:59pm*

2 Introduction

Two-factor authentication (2FA) has become a common approach for authenticating users. Historically, password authentication has been a common authentication approach, but passwords are vulnerable to expose in a variety of ways. so many institutions now apply a second factor of authentication to supplement passwords to prevent attacks due to leaked passwords. 2FA protects the user's account by adding another verification layer to prevent the malicious login even if the user's login credential (e.g., username/password) has already leaked. Authentication questions are often uses as a second factor for authentication. In this project, you will develop a method for creating, managing, and using authentication questions. Note that for simplicity, this project omits the part of username/password creation and verification, only focuses on security login questions, students can assume the username/password is deployed and can always be verified correctly.

Your task will be to implement a series of functionality for the 2FA system. The only file you need to and are allowed to modify is *cmpsc447-user.c*. You will implement the functionality listed below. The detailed description of the tasks that are required to complete this project are in Section 4.

1. **Create a new user - system command interface** - Section 4.2;
2. **User command interface** - Section 4.3;
3. **Question command interface** - Section 4.4;
4. **Command processing from input file** - Section 4.5.

You are encouraged to define other helper functions to modularize your code.

3 2FA Verification via Login Questions

Two-factor authentication (2FA), a type of multi-factor authentication (MFA), is a security process that cross-verifies users with two different forms of identification, most commonly knowledge of an email address, proof of ownership of a mobile phone, and security login questions. Used on top of the regular username/password verification, 2FA bolsters security by making it more difficult for intruders to gain unauthorized access, even if a perpetrator gets past the first authentication step (e.g., leak of username/password).

Today, 2FA is commonly employed in online banking websites, social media platforms and e-commerce sites as a way to harden access controls to the more sensitive areas of a web application (e.g., admin panels or areas that store credit details and/or personal data). Two-factor authentication also enables businesses and public institutions to be more productive and efficient, allowing employees to perform remote tasks with far less security concerns.

Two-factor authentication can play an important role in securing your security credential by blocking a number of application-based attacks. These include brute force and dictionary attacks, in which perpetrators use automated software to generate massive amounts of username/password combinations in an attempt to guess a user's credentials. With 2FA enabled, these attacks are fruitless even if perpetrators are able to discover a user's password, they still lack the second form of identification needed to login to the application.

Additionally, two-factor authentication can help applications counter social engineering attacks, e.g., phishing and spear phishing, which attempt to dupe a user into revealing sensitive data, including their username and password. Even in the event of a successful attack, a perpetrator would still need the additional form of identification required by a 2FA solution.

2FA (or MFA) identification can be categorized into three types:

- **Knowledge factors (something the user knows)** – Common examples are email addresses, username-password combinations, answers to security questions, and the CVV on the back of a credit card.
- **Possession factors (something the user owns)** – Examples of this authentication type include a mobile phone, USB token and a card reader.
- **Inherence factors (something the user is/has)** – This authentication type pertains to unique physical attributes that are inherent to a single person, such as fingerprint readers, retinal scans and voice recognition.

In this project, we will focus on the Knowledge factors by leveraging security login questions to enhance the protection of user's credentials and privacy.

4 Project Requirements

The project executable *cmpsc447-p1* has a simple command line interface: `cmpsc447-p1 <input_pathname> <output_pathname>`. The `<input_pathname>` refers to an input file containing a series of commands to create users, their authentication questions, perform management tasks over those questions, and utilize the questions in authentication. The `<output_pathname>` refers to the output file where print statements that are already in the functions provided will record the progress of your program.

The functions that you implement will extract the commands from the input file (Section 4.1) to create users (Section 4.2) and manage each user's questions. The management of questions is divided between the user-level tasks (Section 4.3) and tasks on individual questions (Section 4.4).

4.1 Project 1 Command Description

See Project 1 Commands below for the description of commands and their functional requirements.

- `user <user_index> <name>`

  Add `user` of `<user_index>` and `<name>` to set of users in the `system_t` data structure `users` (a global) by updating the field `members` and the count in `userct`. Users cannot be deleted once added.

- `add <user_index> <question_index> <question_type> <question> <answer>`

  Add a new question (`<question_type> <question> <answer(s)>`) of `<question_index>` for user of `<user_index>` by adding to the `questions` field in the `user` data structure and updating the count field `qct`. Allocate a new question of appropriate data type (`<question_type>`, either `int_q` or `string_q`) and initialize all the fields in the `question` data structure.

- `remove <user_index> <question_index>`

  Remove question of `<question_index>` for user of `<user_index>`. Remove the question from the `user` data structure (`questions`) and decrement the count. Do not worry about leaving any open spaces in the questions array. You must deallocate the question data structure.

- `change <user_index> <question_index> <question> <answer(s)>`

  Change question (`<question> <answer(s)>`) for `<question_index>` for user of `<user_index>`. Replace the `question` and `answer` fields in the question data structure. We cannot change the "question type".

- `partner <user_index> <question_index_1> <question_index_2>`

  Link `<question_index_1>` and `<question_index_2>` for authentication for user of `<user_index>` using the field `partner`. Both questions must be updated to reflect their partnership. A question can have only one partner, so must update the old partner if a change is made. NOTE: Both questions must be answered if one is chosen at login (authentication) time.

- `login <user_index> <question_index>`

  See if user of `<user_index>` can answer authentication question `<question_index>` correctly. If the question chosen has a partner, then this question and the partner question (i.e., two questions) must be answered successfully to authenticate.

4.2 Create a New User - System Command Interface

To implement the "user" command above, you must implement the function `system_user_new` described below to create a new instance of the `user_t` data structure.

- `user_t *system_user_new( char *user_index_str, char *name )`

  `system_user_new` creates a new user of index `user_index` and name `name`. The input args are (1) `user_index_str` - string for the user index value for the new user, and (2) `name` - name string for the user. The function returns a new user object or NULL upon failure.

4.3 User Command Interface (from cmpsc447-user.h)

Each of the other commands ("add", "remove", "change", "partner", and "login") are implemented by functionality associated with the user and the specific question(s) involved. Here, we list the functions associated with the user (`user_t`) that you must implement and the function pointers in the `user_t` data structure to which they are to be assigned. Where necessary, these functions must invoke the function pointers on the `question_t` data structures as described in the next section. These functions return 0 upon successfully run the corresponding operations, return -1 upon failure.

- `extern int user_add_q( user_t *user, char *question_index, char *question_type, char *question, char *answer)`

- `extern int user_remove_q( user_t *user, char *question_index )`
- `extern int user_change_q( user_t *user, char *question_index, char *question, char *answer )`
- `extern int user_link_q( user_t *user, char *question_index, char *other_question_index )`
- `extern int user_login( user_t *user, char *question_index )`

Note that these functions need to be assigned to the corresponding function pointer fields in the `user_t` data structure.

- `int (*add_q) ( struct user_type *user, char *question_index, char *question_type, char *question, char *answer )`
- `int (*remove_q) ( struct user_type *user, char *question_index )`
- `int (*change_q) ( struct user_type *user, char *question_index, char *question, char *answer )`
- `int (*link_q) ( struct user_type *user, char *question_index, char *other_question_index )`
- `int (*login) ( struct user_type *user, char *question_index )`

These functions must implement the following process:

1. The actions that update the `user_t` data structure (if needed);
2. Find a reference to the appropriate question to update for the user (from the `question_index`);
3. Determine the question type (`int_q or string_q`, see below) from the field `qtype` in the question;
4. Create a reference to the specific question type;
5. Invoke the appropriate (corresponding) function pointer for the question type to perform any updates on the question.

4.4 Question Command Interface (from cmpsc447-user.h)

Most of the complexity in this project comes from the management of questions. First, there are two types of questions: (1) ones with a integer answer and (2) ones with a string answer. The questions will always be in the form of a string. Thus, you must determine the question type from the input (see cmpsc447-user.h for the type indicators), and create questions of the respective types.

To handle the two different types of questions, the project implements a simplified form of object-orientation. There is a `question_t` data type for questions in general. Thus, you can retrieve a question by its index without knowing whether it is an integer question or a string question. However, to operate on the question correctly, you must cast the question object to its appropriate type and invoke the associated function pointer.

Note below that some functions apply to both the string and integer question types. *Also, note that integer questions must always have integral answers, not strings converted to integers.*

Below are the function signatures for the question functions. Note that these functions need to be assigned to the corresponding function pointer fields in the int_q and string_q data structures. These functions return 0 upon successfully run the corresponding operations, return -1 upon failure.

- `extern int question_remove( question_t *q )`
- `extern int question_link( question_t *q, question_t *other )`
- `extern int intq_add( int_q *iq, int qindex, char *question, int answer )`
- `extern int intq_change( int_q *iq, char *question, int answer )`
- `extern int intq_login( int_q *iq )`
- `extern int stringq_add( string_q *sq, int qindex, char *question, char *answer )`
- `extern int stringq_change( string_q *sq, char *question, char *answer )`
- `extern int stringq_login( string_q *sq )`

Specifically, please follow the further requirements below.

- For both `int_q` and `string_q`: Functions `question_remove` and `question_link` are assigned to function pointers `int (*remove)` and `int (*link)`, respectively, of both `int_q` and `string_q`.
- For `int_q` only: Functions `intq_add`, `intq_change`, and `intq_login` are assigned to function pointers `int (*add)`, `int (*change)` and `int (*login)`, respectively, within `int_q`.
- For `string_q` only: Functions `stringq_add`, `stringq_change`, and `string_login` are assigned to function pointers `int (*add)`, `int (*change)` and `int (*login)`, respectively, within `string_q`.

4.5 Command Processing from Input File

- `int main( int argc, char *argv[] )`

`main` runs the sequence of commands provided in `<input_file>` and stores the results of the commands in `<output_file>`. The output `fprintf` statements are provided. The input args are (1)`argc` - The number of command line arguments (which is 3), and (2)`argv` - Command line arguments (`cmpsc447-p1 <input_file> <output_file>`).

Students are required to complete three fragments of this function to: (1) get an input line (i.e., the next command) from the input file; (2) compute a reference to the `<user_index>` string to pass to the `find_user` function from the input line; and (3) compute a reference to the arguments list for `apply_command`. These location where these three tasks are required students should complete the `TBD(x)` functionality in the `main` function for these three cases.

5 Doing the Project

Perform the following tasks to complete the project.

5.1 Download and Install Virtual Machine

- Download VirtualBox 6.1.30 from the link: https://www.virtualbox.org/wiki/Downloads.
- Download the pre-configured Ubuntu 20.04 Virtual Machine from the link: `https://drive.google.com/file/d/1pDE9lHg6vyBZO7kaS79FKgZ-OkH7B5TO/view?usp=sharing`
- Launch VirtualBox, select "Import", and then choose the Ubuntu 20.04 VM image that you downloaded from above, then click OK to let the VM be imported.
- In order to let the VM be able to connect to the internet, we should open Oracle VM VirtualBox Manager, then click on File - Preference - Network - Add New NAT Network - OK.
- Username is cmpsc447, password is 123456.
- If you want to create a shared folder with host, goto Settings - Shared Folder to create a folder on the host, then run `sudo adduser your_username vboxsf`.
- Note that the guest-additions and enhancement are already installed in VituralBox, so you can copy from the VM using `Ctrl+Shift+C` and paste to the VM using `Ctrl+Shift+V`.

5.2 Download Tarball

The project tarball is provided along with these instructions in the Canvas assignment.

5.3 Project Tasks

You will then perform the following steps.

- **Task 1**: Download VirtualBox and VM, install the VM and deploy the code base of the project.
- **Task 2**: Develop the functions introduced in Section 4. You may choose to write subroutines if appropriate. You are only allowed to insert your own code in cmpsc447-user.c, do not change any existing code, including existing functions, prototypes of the functions to be implemented, and data structures as well as macros in cmpsc447-user.h.
- **Task 3**: Test your own code using the provided test cases per the instructions below. Feel free to design your own test cases, but you do not need to hand those in.

6 Testing

We will test your submission using the same environment as the provided Ubuntu 20.04 VM image, please develop and test your project using the provided VM to ensure your code works on it.

We will test your program by supplying the three input files provided in the project tarball (`test, test2, test3`). `test` is the most basic test case for simple changes, removals, and partnering of questions. `test2` extends on these test in more complex conditions to determine whether the program works as expected when questions are recreated and partners are removed. `test3` assesses the program's operation under a variety of error cases. The expected output is shown in the three corresponding output files (`out, out2, out3`).

We will provide sample test input files in the project tarball, please refer to the sample test input and output files in the project tarball to understand the desired functionality. We will test for these cases specifically

in this project, although we may decompose the test cases for ourselves to avoid cascading errors. Also, we reserve the right to create different, additional test cases, although any additional test cases will test the functionality described above.

Again, You should use the provided VM to verify that your code works. We developed and tested the project code on the VM, so should work fine, but it is up to you to make sure.

7 Deliverables

Submit a tarball containing your code and the output of the test cases as described below. We expect that the code has been tested in the provided VM, so points will be deducted if the code fails in the VM for some minor compatibility reason.

Create the output files using the names `submit-out` for test, `submit-out2` for test2, and `submit-out3` for test3. Then build the submission tarball using "make tar." This command is already designed to capture your code and the output files, if named in this manner.

8 Grading

The project will be graded based on the following:

1. **Logistics**: The project submission is complete. In addition, the project builds and runs on the provided VM without compatibility issues (10pts)
2. **Add New User**: New users are added and always available (10pts)
3. **Add New Question**: New integer and string questions are added correctly and can be used in authentication individually (30pts)
4. **Question Changes and Removal**: Question changes and removal are implemented correctly for string and integer questions (20pts)
5. **Partnering Questions**: Partnering questions should work as expected for authentication under question changes/removal (30pts)