CS 465 Cybersecurity Principles and Practice
Programming Assignment #2

Assigned: Thursday, April 4, 2019
Deadline for submitting the assignments electronically: 12 noon on Thursday, April 18, 2019
Deadline for the hard copy: beginning of the class on Thursday, April 18, 2019
No late assignments

Have you ever been bothered by your grandmother attempting to friend you on Facebook? Have your parents been less than thrilled by some "fun" picture you or your friend posted on Facebook? Then this assignment is for you. (Even if you answered "No" on these questions you still have to complete the assignment.)

The goal of this assignment is to implement access control to a simplified Facebook, called MyFacebook. The system consists of only one profile owner (you), who has friends that can belong to zero, one or more lists. A person has to be your friend to view your profile and post pictures to your profile. You, the profile owner, are able to perform administrative tasks such as adding friends, creating lists, assigning friends to lists, and changing owners and setting the access control of pictures posted on your wall. In other words, you are the administrator (or root user) of your profile.

In MyFacebook if your friend John posts a picture on your wall, by default he is the owner of the picture. However, you may not want your friend John to dictate who can or cannot see the picture he has posted on your wall. Since you have administrative privileges over your profile, you can change the owner of any picture which is posted on your wall (i.e., you can make another friend be the owner or you can become the owner). Every picture that is posted on your profile has an ACL associated with it, that is, it has an owner, a list associated with it, which is optional, and a tuple of access permissions. As any trusted system, your system will log its activity in an audit log file.

The programming assignment does not include the GUI. You will only implement the functionality behind the access control of your profile. The input will be given in an ASCII file, which contains the instructions to be executed.

Your program must compile and run on the LCSEE Ubuntu Linux *shell* servers. The programming assignment may be implemented using any of the following compilers or interpreters:

- gcc (Ubuntu 5.4.0-6ubuntu1~16.04.5) 5.4.0 20160609
- Java 8 (openjdk 1.8.0_151)
- Python 2.7.12 & 3.5.2

The program executable must be named `access`, or the closest possible equivalent depending on the programming language that you chose. For example if you use Java, the file name for the class containing the main method should be `access.class`. The program will be invoked by the following command line (or equivalent depending on programming language),

```
./access filename
```

where `filename` is a command line argument that stands for the name of an ASCII file containing the instructions that the program should interpret. An example of this could be:

```
./access testcase1.txt
```

The program must accept the name of a text file as input via a command line argument and respond appropriately based on a predetermined set of instructions in the file. Each instruction will be written on a separate line. It can be assumed that all instructions will be free from any syntax errors, that is, each line in the input file will contain a syntactically correct instruction.

The following is the list of instructions:

`friendadd` *`friendname`*

`viewby` *`friendname`*

`logout`

`listadd` *`listname`*

`friendlist` *`friendname listname`*

`postpicture` *`picturename`*

`chlst` *`picturename`* `list`*`name`*

`chmod` *`picturename rw rw rw`*

`chown` *`picturename friendname`*

`readcomments picture`*`name`*

`writecomments` *`picturename text`*

`end`

In order to create your MyFacebook profile, the very first line of the instructions file has to be of the form:

```
friendadd profileownername
```

This will be the ONLY time when the `friendadd` command can be executed without anybody, including yourself, viewing your profile. If the very first command is not of the form described above, then your program should report an appropriate error (both to the console and audit log) and terminate. After execution of this command, you (the profile owner) need to view your profile to perform any further actions. Once you view your profile, you will have administrative privileges and will be able to issue any valid command. Remember that access permissions do not apply to the profile owner.

A `friendname`, `listname`, or `picturename` may consist of up to 30 ASCII characters except for a forward slash ('/'), colon (':'), or any of the following white space characters: carriage return, form feed, horizontal tab, new line, vertical tab, and space. Instructions, friend names, list names, and picture names are all case sensitive.

A list of friends should be maintained in an ASCII text file called `friends.txt.` You may organize this file any way that you like as long as it is used by your program for friend authentication AND you clearly and thoroughly explain your approach and the format of your `friends.txt` file in your program documentation. Authentication is performed when somebody attempts to view your profile.

A friend may be a member of zero or more lists. Your program should use an in-memory data structure to keep track of friend lists. After all instructions have been processed, but prior to terminating, your program must create a file named `lists.txt`. The `lists.txt` file should contain a listing of all lists and friends who belong to each list. The exact file structure will be provided as part of example test cases.

Each picture that you or your friends post on your profile is associated with an owner, at most one list of friends and a set of permissions. The friend who posted the picture to your profile is by default the owner of that picture. To simplify the assignment each picture will be represented by a text file. Thus, for each picture posted on your profile with the command `postpicture`, your program must create a physical text file and provide the ability to read (comments) from and write (comments) to the file. The file should be created in the same directory as the program executable (i.e. the "current" directory). Your program should use an in-memory data structure to keep track of pictures created and their corresponding access rights. After all instructions have been processed, but prior to terminating, your program must create a file named `pictures.txt`. The `pictures.txt` file should contain a listing of each picture posted on your profile i.e. picture name, picture owner, associated list of friends, and the tuple of permissions. The exact file structure will be provided as part of example test cases.

Upon the execution of each instruction, the program should DISPLAY an appropriate comment on the screen AND LOG the SAME comment into an audit log file called `audit.txt`.

Friends, lists, pictures, and permissions do not need to persist across multiple executions of your program. That is, each execution of your program should start with a "clean slate". (The `friends.txt`, `lists.txt`, `pictures.txt,` and `audit.txt` files should be completely overwritten if they already existed when your program first started. This is also the case with any files that were created via the `postpicture` command. If a file already exists when your program starts, its content should be overwritten. However, subsequent calls to the `writecomment` command that are issued during a single execution of your program should append the corresponding comments to the appropriate file.

Detailed specifications of commands

`friendadd friend`***`name`***

- Example: `friendadd tom`
- Creates a friend with a specified name.
- By default, this friend should **NOT** be a member of any list
- Friend's names are stored in an ASCII file called `friends.txt` for later use (Your program has to reference this file for user authentication when executing the `viewby` instruction).
- Things to consider / constraints:
  - Two friends with the same name CANNOT be created
  - This instruction may only be executed by the profile owner. The only exception is the first instruction which creates the profile owner, as previously discussed.

`viewby` ***`friendname`***

- Example: `viewby tom`
- Checks friend name against records in the `friends.txt` file. If the friend name exists, this person can view your profile. Otherwise, an error message is displayed & logged, and your friend is not allowed to view your profile.
  - You (the profile owner) must also view your profile first, in order execute any other instruction. (The only exception is the first instruction which creates the profile owner, as previously discussed.)
- Things to consider / constraints:
  - To simplify the assignment MyFacebook does not support concurrent users. That is, only one friend at a time can view your profile. (If one of your friends is viewing your profile, another friend cannot view it.)
  - Once a friend views your profile, he/she will be able to use the `postpicture`, `readcomment`, `writecomment`, `chlist`, `chmod`, and `logout` instructions. The `readcomment`, and `writecomment` access is allowed/not allowed to friends according to the picture access rights found in `pictures.txt` file.
  - There MUST be somebody (i.e., profile owner or a friend) viewing your profile in order to be able to use any other command.

`logout`

- Example: `logout`
- A friend or you no longer views your profile.

`listadd` ***`listname`***

- Example: `listadd parents`
- Creates a list to which friends may be added. (Note: A list may be associated with a picture using the `chlst` instruction explained later in this document.)
- Your program should use a data structure to maintain a listing of all lists that have been created and the friend(s) that belong to each list.
- Things to consider / constraints:
  - This command can only be executed by the profile owner.
  - A list is initially created without any friends belonging to it.
  - Two lists with the same name CANNOT be created
  - A list may not have the name `nil`. (`nil` is reserved for the cases when no group is associated with a picture.)

`friendlist friend`***`name`*** `list`***`name`***

- Example: `friendlist tom parents`
- Adds the specified friend to the specified list.
- Things to consider / constraints:
  - A friend can be associated with multiple lists
  - If the specified friend or list does not exist, then an appropriate error message should be displayed and logged.
  - This command can be executed only by the profile owner.

`postpicture` ***`picturename`***

- Example: `postpicture picturename.txt`
- This command emulates posting of a picture to your profile by a friend i.e. it creates a text file (i.e. picture) with default permissions (`rw -- --`). The first line of this textual file should be the name of the picture (i.e. `picturename`)
- Your program should use a data structure to keep track of the picture name, owner, list, and permissions.
- Things to consider / constraints:
  - Two pictures with the same name CANNOT be created
  - The profile owner or a friend must be viewing the profile in order to execute this command
  - The owner of the file should be set to the person currently viewing your profile and issued the `postpicture` command. (The owner can be changed by the `chown` command, as described later.)
  - Initially no list is associated with the picture (i.e., its list should be `nil`)
  - The default permissions for the picture should be as follows: `rw -- --`

`chlst` ***`picturename listname`***

- Example: `chlst picturename.txt parents`
- Changes the list associated with a file.
- Things to consider / constraints:
  - A friend or you must be viewing your profile in order to execute this command. To simplify the assignment, only a single list can be associated with a picture.
  - If you want to disassociate a picture from all groups, then `nil` should be used for *`listname`*.
  - This command can be executed by the current owner of the picture or by the profile owner.
  - The list for a picture can only be changed to `nil` or a list to which the owner belongs. For example, suppose there are three lists `A`, `B`, and `C`. Now suppose that friend `X` owns picture.txt and belongs to lists `A` and `B`. Friend `X` would be able to issue the commands `chlst picture1.txt A`, and `chlst picture1.txt B`, but NOT `chlst picture1.txt C`. This restriction does not apply to the profile owner. The profile owner may change the list associated with a file to any valid list.
  - If you do not want to have friends deciding for you who can/cannot see a picture on your profile that they have posted, you should change the owner of the picture. Keep in mind that your friend has to stop viewing your profile and then you must view your profile in order to issue this command.
  - If the specified picture or list does not exist, then an appropriate error message should be displayed and logged.

`chmod` ***`picturename rw rw rw`***

- Example: `chmod picturename.txt rw r- --`
- Changes the access permissions associated with a picture.
- Your program does not need to physically manipulate the picture permissions, but it should maintain permission consistently via a data structure.
- The first `rw` parameter indicates the permissions for the owner of the file. The second `rw` parameter indicates the permissions for the list with which the file is associated. The third `rw` parameter indicates the permissions for others (i.e., friends who are not owners of the file, or who do not belong to the list associated with the file).
- Permissions are specified via a two letter string consisting of the following characters: `r`, `w`, and `-`. The first letter of a permission string can be `r` or `-`. An `r` indicates that read access to the comments in the picture is granted, while an `-` indicates that read access is denied. The second letter of a permission string can be `w` or `-`. A `w` indicates that write comments (to the picture) access is granted, while an `-` indicates that write access is denied.
- Things to consider / constraints:
  - A friend must be viewing your profile in order to execute this command
  - This command can be executed by either the current owner of the picture or by the profile owner. This means that unless you change the owner of the picture to be you (the profile owner), the friend that posted this picture can determine who can read and write comments on it.
  - If the specified picture does not exist then an appropriate error message should be displayed and logged.

`chown` *`picturename friendname`*

- Example: `chown picturename.txt tom`
- Changes the owner of a file.
- Things to consider / constraints:
  - This command can only be executed by the profile owner.
  - If the specified picture or friendname does not exist, then an appropriate error message should be displayed and logged.

`readcomments` *`picturename`*

- Example: `readcomments picture.txt`
- Allows viewing the picture and displays the comments associated with the picture (i.e., it displays the first line in the `picture.txt` file, which is the name of the picture and all the other lines, i.e., all comments)
- Access to this command is granted or denied according by the following security model:
  - If the current friend who is viewing your profile is the owner of the picture and read access for owner has been granted, then the instruction may be carried out.
  - Else if the current friend who is viewing your profile is not the owner, but he/she is a member of the list which is associated with that file, and read access for the list has been granted then the instruction may be carried out.
  - Else if the current friend who is viewing your profile is not the owner, he/she is not a member of the same list that is associated with the file, and read access has been granted for others, then the instruction may be carried out.
  - Otherwise access should be denied.
- Things to consider / constraints:
  - If the picture does not exist, then an appropriate error message should be displayed and logged.
  - A friend must be viewing your profile in order to execute this command

`writecomments` *`picturename text`*

- Example: `writecomments picturename.txt sometext`
- Allows viewing the picture and adding a comment to a picture by appending `sometext` to the `picture.txt`
- Access to this command is granted or denied according by the following security model:
  - If the current friend who is viewing your profile is the owner of the picture and write access for owner has been granted, then the instruction may be carried out.
  - Else if the current friend who is viewing your profile is not the owner, but he/she is a member of the list which is associated with the file, and write access for that list has been granted then the instruction may be carried out.
  - Else if the current friend who is viewing your profile is not the owner, he/she is not a member of the list which is associated with the file, and write access has been granted for others, then the instruction may be carried out.
  - Otherwise access should be denied.

- Things to consider / constraints:
    - If the picture does not exist, then an appropriate error message should be displayed and logged.
    - A friend must be viewing your profile in order to execute this command
    - The text should be APPENDED as a new line to the end of the file. Make sure your program does not totally overwrite the file.

end

- Example: `end`
- The last instruction in the input file.
- Indicates that your program should write out the `friends.txt`, `lists.txt`, `pictures.txt,` and `audit.txt` files as described previously, perform any other necessary cleanup actions (i.e., closing open file streams) and terminate.