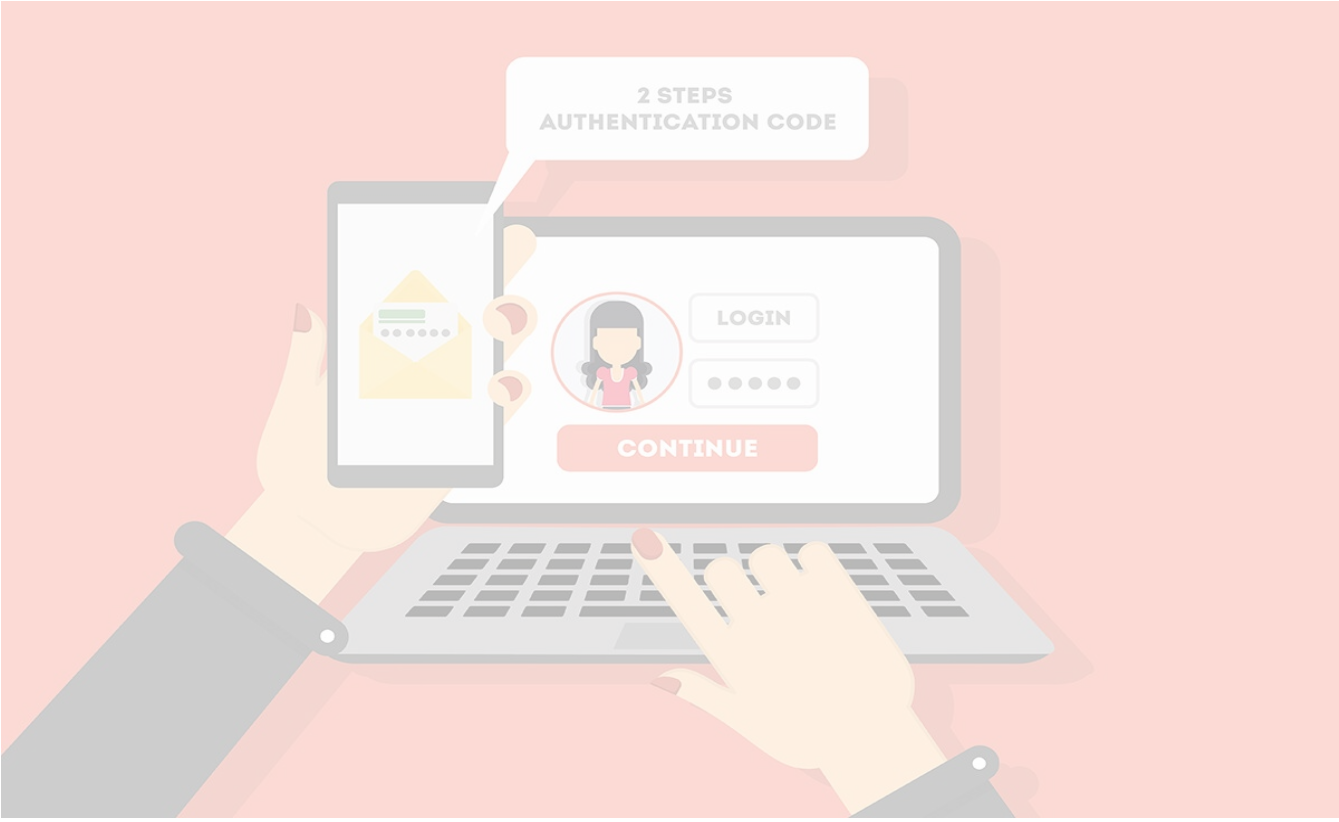
2-Step Authentication



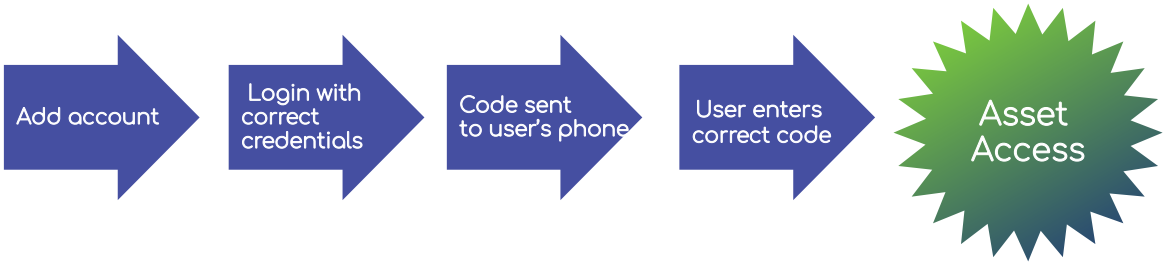
Tuyet Pham
July 20, 2019
University of North Texas Engineering Dep.
Dr. Pradhumna Shrestha

Introduction

On the morning of July 7th 2019, I received a phone call from my dearest mother. She exclaimed to me that her Facebook had gotten hacked the night prior. Being in the field of technology, I had offered my help. Realistically I was not going to be able to retrieve her Facebook account, but I did however change her email password to something other then her name and up her security measures to prevent future attacks. Besides from the password that had contained her name, I noticed she did not set up any of additional verification nor did she sign up for the 2-step authentication process. Three hours into the enjoyable process of purging spam and setting further security measures for her other accounts, I couldn't help but to wonder of how this could have been prevented with just the 2-step authentication procedure alone. Perhaps if that was set up, the hacker would have given up and picked someone elses' afternoon to ruin. For this reason I chose to go with setting up a 2-step authentication procedure within my program.

Two step authentication (also known as 2FA) is a type of multi-factor authentication. It is the process of confirming that 'who you are' is 'who you say you are', by 2 different factors. In more common cases this means it is a password (factor 1) and a one-time code (factor 2) that is generated, sent to an authenticator such as a smart phone, email etc.. that only the user possesses. Depending on the level of sensitivity of assets, more than 2-factors can be involved to authenticating a user. Advances in research and improvement in hardware, such as GPS or microphone, could lead to other factors potentially being involved in this process.

Apart from the coincidental attack on my mother's Facebook account, I chose this project because I've personally never worked with APIs or SMS. The program displays basic functionality of a 2-Step authentication process. The user will first have to create his/her login information. This pertains to a username, password and a phone number to send the one-time-code to. After the account is created the user can now log into the newly created account. Once they give the correct username and password, the one-time-code will be sent to their phone and the program will prompt them for the code to verify that it is indeed them.



This 2-Step authentication process can be applied to a multitude of applications such as logging into a bank accounts, email accounts, social media accounts and many others. The need for this level of security is no longer questionable. Technological advances open

more doors for opportunities than it seems to close. Although this is a great thing, if we are not careful, the wrong people may go through those doors uninvited.

Methodology

Language

The languages used for this project are C++ and Python. Only C++ was chosen initially, but due to complications in personally finding well documented, and free messaging API for C++, Python was added to aid the messaging process using the Twilio SMS API. The Python script will be invoked within the C++ application, and will return exit(5) for failure.

Design

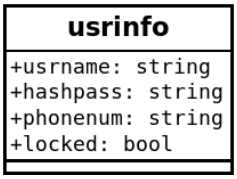
Prerequisite

The program requires you to have python2.7+. Python3.6 is recommended. The program will be using the Twilio messaging API, which requires an account made on twilio web site and services. A python script from twilio was provided as a layout to use their Client object. The Twilio API require a client's SID, token, client's phone number, message and contact number. The Twilio API can be download using pip, but pip3 is recommended.

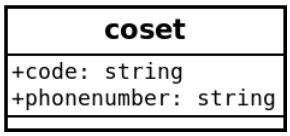
Structurally, the program is fairly simple. The main functionalities broke down into 3 parts; the ability to add an account, log into to that account, and then verify the one-time-code that is sent to your phone. The required files for the program to run are '.twiliosms.py' and the '.passwdlist.txt' which are located in the etc folder. The passwdlist.txt will be the main database location for users information. It will store three attributes for each user. The three attributes are the username, hashed password, and their associated phone number. Each attributes are deliminated by a semicolon, and each user will occupy a single line or row. I will continue to refer to this file as the database. The database will be made by the program if it is not initially there already. If for any reason an external database file must to be used the file will have to comply with the same format and be renamed to '.passwdlist.txt'. The external database will also have to comply with the program's hashing standard. At this point there are no functionality to hash external files to comply with the internal hashing algorithm, so this is definitely not recommended. Along with these 2 important files all headers and sources are required as well. This program is also native to Linux and Unix systems.

Objects and Structures

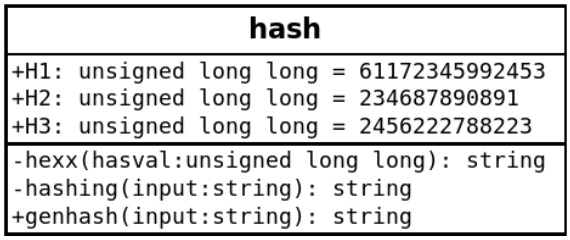
After some thoughts, the conclusion was to create more structures than objects. C++11 allows private access modifiers for structure as it does for classes. This feature made it abundantly easier to have object like structures thus making it less complicated. The structures are 'usrinfo', 'coset', and 'hash'. All structure definitions can be located in 'resources.h'. The only class in this program is 'sms', and the definition will be in it's corresponding header and source file.



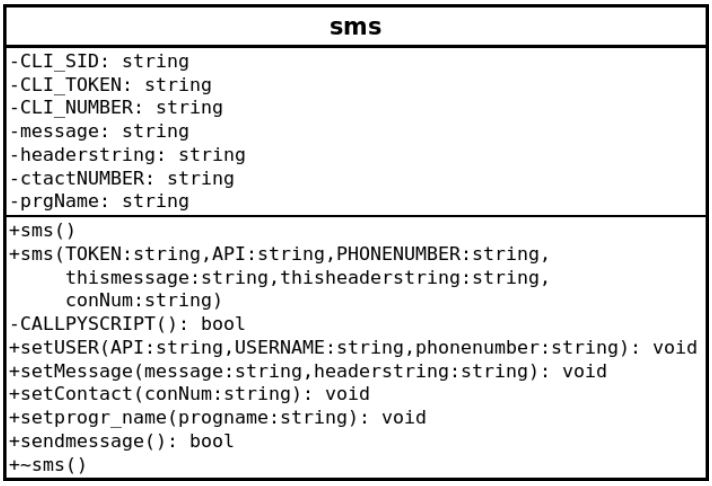
Drawing 1: usrinfo structure



Drawing 2: coset structure



Drawing 3: hash structure



Drawing 4: sms class

Usrinfo structure

The structure usrinfo is used to load the information of users that is pulled from the database. The usrinfo is then stored into a stl vector named oldu, which stores pointers to usrinfo structures. Looking at *Drawning 1* the usrinfo structure stores, as strings variables, username, hashed password, and phone number. There is also a boolean variable named locked, which was unused by the program. The purpose or idea of this variable is to lock the user out after 3 attempts.

Coset structure

The structure coset is used to store the code that was made after the user verify their username and password, and the user's phone number that the code was sent to. The coset structure is stored into a stl vector named codes, which stores pointers to coset structures. This is to compare to what the user input is when they are asked for the code. This will also make sure that the code is coming from the correct phone number.

Hash structure & it's algorithm

The structure hash is used to hash the password chosen by the user when creating their account. This structure is used by calling it's only public function **`genhash()`**, and giving it a string that should be the user's password. The genhash's only responsibility is to throw the string to the hashing function named **`hashing()`**. The hashing function has a local variable named hashval of type unsigned long long. The function will iterate through each char of the string given and OR(|) hashval with that char, then placing it in hashval. The hashval will then multiply itself with H1 at value 61172345992453, H2 at value 234687890891, and H3 at value 2456222788223. These numbers were picked at random and are not prime. After the iteration is done the hasval is then thrown to function **`hexx()`** to get turn into it's hexadecimal value by using std::stringstream from C++ library sstream, and the std::hex I/O basefield modifier. Finally everything gets sent down the stack and is return through the gethash() function to be stored into the database.

Sms class

The sms object is used only after the user verifies their credentials. This means the user had to provide the correct username and password. The overall job of the class is to use the information provided such as client token, client sid, client phone number, message, header message, contact number (user's phone number), and the name of the Twilio python script, to send the user the one-time-code.

The class has two constructors; one without any parameters, and one with all the parameters. The class requires all arguments to be met before the script is ran. Failure to do so will alert an error message coming from the python script. Other aiding setting functions are listed below.

- `void setUSER()`
- `void setMessage()`
- `void setContact()`
- `void setprogr_name()`

After setting all information needed to send the message, the public function **`sendMessage()`** should be called to invoke the sending of the actual message. This function will return false upon failure. Within the class this means the **`sendMessage()`** will call the private **`CALLPYSCRIPT()`** function, which will then to do a system call with all parameters in additions to the path of the Twilio python script. The python script will exit(5) for failure, if noncompliance with the arguments quantity. This will send a boolean of false down the stack.

This main purpose of this class is to compartmentalize the messaging functionality from the main program to prevent further issues. Personally I thought this was the hardest part and by doing it this way I was able to do everything else, the storing and the interface, first.

Main program

The main.cpp file is very crowded with several smaller functions that are very simple in terms of functionality. Most of these functions acts as client side mediation. All function declaration for main.cpp can be found in resources.h. All functions will be listed below for references. I will only elaborate in details the more important ones in this documentation.

- `int MMENU()`
- `bool ADDLOGIN()`
- `bool USRCHECK_R(std::string)`
- `bool PHONECHECK_R(std::string)`
- `bool USRCHECK_F(std::string)`
- `bool PHONECHECK_F(std::string)`
- `std::string GETPASS()`
- `bool SAVETO(usrinfo * thisu)`
- `bool LOGIN()`
- `bool USERCREDCHECK(std::string username, std::string password)`
- `std::string GENCODE()`
- `void DELCODE(std::string phonenumber)`
- `bool SENDSMS(std::string phonenumber)`
- `bool RECIEVER(std::string phonenumber)`
- `bool CHECK4LIST()`
- `bool CREATELIST()`
- `bool CHECKTWILIO()`
- `bool LOADTO(std::string filename)`
- `void LOADTOAUX(std::string line)`
- `bool FREEALL()`
- `void FRONTPAGE()`

The initial thing the program will do is check for the existence of the password list (the database) and Twilio python script by calling the **`CHECK4LIST()`** function and the **`CHECKTWILIO()`** function. If the program detects that there is no database in the designated folder, it will create one with the **`CREATELIST()`** function. There is no way around the Twilio python script, the program can not run without this script

After verifying the two file's existence the program can now load in the existing database using the **`LOADTO()`** and **`LOADTOAUX()`** functions. These functions will take each line (row) of the file, create a new usrinfo structure and append the usrinfo to the vector oldu. Finally after loading the database, the user can now Login, Add account or Quit the program. Of course if the user has never made an account they can not log in.

The login process involve functions

- `bool USERCREDCHECK_R(std::string, std::string)`
- `std::string GENCODE()`
- `bool RECIEVER(std::string phonenumber)`
- `void DELCODE(std::string phonenumber)`
- `bool SENDSMS(std::string phonenumber)`

The **`USERCREDCHECK_R()`** will check if the username and password given matches what is in the database. Once the verification is successful the program will generate a code using the **`GENCODE()`** function. The code is a seeded random integer from 50,000 to 100,000. After code generation the **`RECIEVER()`** function is called to (1) call the **`SENDSMS()`** function and (2) check if the code sent matches the code given. The

SENDSMS() function will instantiate a new sms object named autho, set all it's criteria and call the **sendMessage()** class function. Once the script ran successfully it will return true down the stack and the **RECIEVER()** function will wait for the user's input. Once the user input the correct one-time-code the code will be deleted from the codes vector.

The add login process will involve functions

- `bool ADDLOGIN()`
- `bool USRCHECK_R(std::string)`
- `bool PHONECHECK_R(std::string)`
- `bool USRCHECK_F(std::string)`
- `bool PHONECHECK_F(std::string)`
- `std::string GETPASS()`
- `bool SAVETO(usrinfo * thisu)`

The **ADDLOGIN()** function declares a hash structure name hasher and will use it's member function **genhash()** to create the hashed password for the user once it is verified that this user's credential has the correct formatting and is not using a repeated phone number or username. These client side mediation functions are **USRCHECK_R(), USRCHECK_F(), PHONECHECK_R(),** and **PHONECHECK_F().** Once the formatting and repeating checks are done the user will be prompted by the **GETPASS()** function to input a password, once done the function will return the password string. Next the password string will be thrown to the member function **genhash()** of the hasher variable to get the hashed password. The newly created usrinfo pointer, named thisu, is pushed back to the oldu vector, as well as sent to **SAVETO()** to be save into the physical (passwdlist.txt) database file.

Results

The program works by taking advantage of simple user error checking, the Twilio messaging API, and a simple .txt file that's acting as a database for user's information. The person using the program should keep in mind that every files needs to be in place before running, the program. The program relies on the .twiliosms.py and the passwdlist.txt in order to run. Without a database the program cannot create new and or use login information.

Option 1 | The Login

In order to login the program will need you to create an account first. If you have not already made an account, please do so by choosing option 2 at the main menu. If the passwdlist.txt does not exist, the program will make you one. In any case you want to start a new file (lets say you want to have a list for each types of accounts), you or your administrator will want to move the list to a new directory and rename it. The program will again make a new list for you. This can be a continual thing.

Once the database is populated the users can login to their designated account. The program will prompt for the user's username and password. Once those credentials are

met a message containing the one-time-code will be sent to the phone number that the user have on file. After a successful login the program will end.

Illustration 1: Main menu

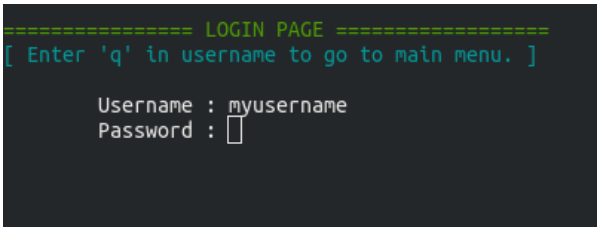


Illustration 2: Login Page

**Note that the password will not be visible.*

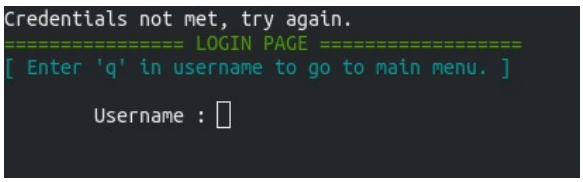


Illustration 4: If credentials are not met

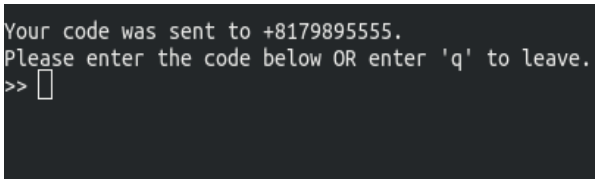


Illustration 3: After credentials are met

Option 2 | Add Account

In order to login you will have to make an account. For this, option 2 will need to be chosen initially. First the program will prompt the user to enter their desire login without any space. The program's formatting will deny any username that doesn't comply. After the username the program will ask for the user's phone number. Since these two attributes are the most critical attributes, the program will error check first before moving on to the password portion. After all error checking on the new user's information is pass, the user will then be prompt for the password twice. The password currently has no error checking and can store any string including spaces.

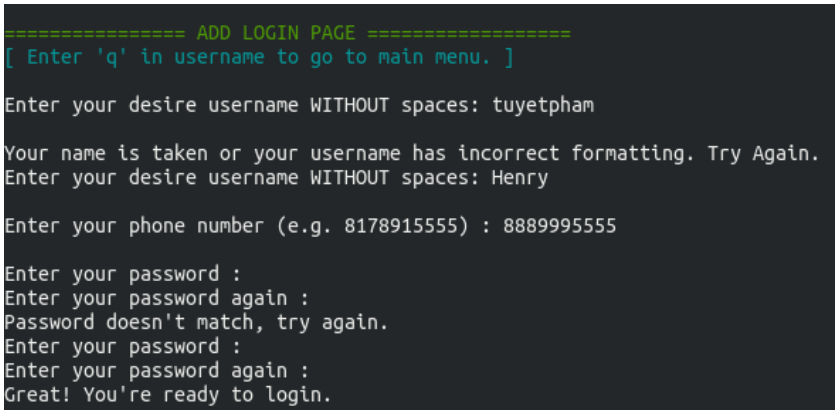


Illustration 5: Add account page

**Note that the password will not be visible.*

If the user did everything correctly their information will be stored into the database. You can check this by looking at the passwdlist.txt file. The password will be hashed so it will not be in plain text. This is also a one-way hash so it can not be backward hashed. No deletion functionality is available at the moment.

To quit the program, simply choose option 3 at the main menu page.

Conclusion

I chose this project because I have personally never done anything related to messaging. Learning about the technical process of the messaging API and scouting out a provider was challenging but also a very great learning experience. On the non-technical side, I have learned that documentation is very extremely important because although someone provider seems to have a very good services, they lack documentations for other developer to use their libraries. I have also learn that C++ is a very hard language to link to newer API SMS libraries, as those seems to favor more scripts and higher languages. This however had made me more appreciative of the python language as it is pretty easier to work with.

If I had more time, I would want to implement further error checking for users credentials. I could add more restriction on the passwords and more of the server side mediation in the sms class. I would also spend sometime learning the full potential of the Twilio SMS API and perhaps implement other types of functionalities such as chat. This is not required but it is fun to think about.