

# Assignment – C Lab (Safe Storage)

## Introduction and Setup

The task focuses on a defensive coding style, security-oriented, and avoiding potential vulnerabilities. The implementation must follow the instructions given in the lectures and laboratories.

- Visual Studio Community installation kit: [Visual Studio Community](#). It is recommended to use the latest version, **Visual Studio Community 2022**.
- You must compile with **Warning Level 4** and **Treat Warnings as Errors: YES** (*right-click on the project in Visual Studio – Properties – C/C++ – General – Warning Level – Treat Warnings as Errors*).
- You must ensure **proper memory management** and release all allocated resources appropriately. You can check out the following link: [Find memory leaks with the CRT library](#).
- You will deliver a zip file named **LastnameFirstname.zip**.
- For development, use Windows APIs (e.g. **CreateThread/ WaitForSingleObject/ CreateFile/ ReadFile** etc.). Documentation can be found on MSDN (example: [CreateFileA function](#)).
- For bonus points, you can also use **SAL (Static Code Analyzer)** – [Understanding SAL](#).

## Project Structure

```
[SafeStorage]
|- [SafeStorage]
    |- main.c
|- [SafeStorageLib]
    |- Commands.h
    |- Commands.c
    |- includes.h
|- [SafeStorageUnitTests]
    |- SafeStorageUnitTests.cpp
    |- test_includes.hpp
|- SafeStorage.sln
```

You will open the file **SafeStorage.sln** after installing Visual Studio. You will notice that there are two projects, **SafeStorage** and **SafeStorageLib**:

- **SafeStorageLib** is a static library where you will implement all the functionality of the project. All the code you deliver will be in **SafeStorageLib**. You must follow the provided interface and the instructions in the headers. All newly created files will be in this project.
- **SafeStorage** is a console application that has a minimalist command-line parser and links with the static library **SafeStorageLib**. There are several vulnerabilities present in the code. You will deliver a text file named **vulnerabilities.txt**, where you will indicate the vulnerabilities found, the line where the vulnerability appears, and how it can be fixed.
- **SafeStorageUnitTests** is an unit-test project where more functionality can be added by you to test your solution. Here you can simulate brute force attacks, fuzzing, and other scenarios you wish.

## Requirements

You are required to implement a console application (**SafeStorage.exe**) that simulates a platform where students can upload their assignments. This application will run with administrator privileges from its directory (which we will refer to as **%AppDir%**) and will expose several commands (the command names are **lowercase**). In the **%AppDir%** directory, the application must create and manage at least the following two resources:

1. a **users** subdirectory (which will be accessible to users) – **%AppDir%\users**
2. a **users.txt** file (which simulates a database with the credentials of all registered users – username & password)

Optionally, you can create other files/ resources in the **%AppDir%** directory if needed.

To see how you can obtain the current directory **%AppDir%**: [GetCurrentDirectory function](#).

Available commands (see **Commands.h** file for more details):

### 1.register <username> <password>

- After each successful registration, a subdirectory will be created in the **%AppDir%\users** directory with the name of the newly registered user. For example, the command **register Alex Password1@** will create the subdirectory **%AppDir%\users\Alex** and will add a new entry in the **%AppDir%\users.txt** file for user **Alex** and their password.
- If the same user tries to register two or more times, an error status will be returned.
- The user's password will be stored securely. In the **%AppDir%\users.txt** file, **the password will not appear in plain text** – you will need to encrypt it, create a hash over the password... – this is up to you.

- **This command is available only if no user is logged in; otherwise, an error status will be returned.**
- Each user will only have access to their own subdirectory – user **Alex** will not be able to see the directory of user **Andrew**.
- You must sanitize the input:
  - A username is considered valid if it contains only English alphabet letters (a-zA-Z) and is between 5 and 10 characters long.
  - The password must have at least 5 characters and contain at least one digit, one lowercase letter, one uppercase letter, and at least one special symbol (!@#\$%^&).
  - Other characters are considered invalid, and an error status must be returned if encountered.
- To see how you can calculate a hash using Windows APIs, you can visit the following link: [Creating an MD-5 hash from file content](#).

## 2. login <username> <password>

- **This command is available only if no user is logged in; otherwise, an error status will be returned.**
- The `%AppDir%\users.txt` file will be opened, and the user's credentials will be verified.
- After the successful execution of this command, the `store` and `retrieve` commands can be executed (see below).
- **You must take protective measures against brute-force attacks!** *Suggestion: count how many failed login attempts occurred in the last second, and if it exceeds a certain threshold (e.g. 5 attempts), do not allow any more logins, even if the correct credentials are provided!*

## 3. logout

- **This command has no parameters.**
- **If no user is logged in, the command fails.**
- After the execution of this command, we can `login` again.

## 4. store <source\_file\_path> <submission\_name>

- **This command DOES NOT work if no user is logged in (an error status will be returned).**
- The `store` command will read the file specified by `<source_file_path>` and copy its contents to `%AppDir%\users\<current_user>\<submission_name>`.
- Assuming user **Alex** is already logged in and executes the command `store d:\\homework_mark10.txt homework1`, this will result in copying the contents of the file `d:\\homework_mark10.txt` to `%AppDir%\users\Alex\homework1`.
- Files can have a maximum size of **8 GB** – you must use a thread pool with a fixed number of **4 threads** and transfer the files in fixed-size chunks (any size between **512 B** and **64 KB**). You have two options: either write your own thread pool or use the existing one in Windows [Using the Thread Pool Functions](#).
- **Pay special attention to synchronization issues!**

## 5.retrieve <submission\_name> <destination\_file\_path>

- This command **DOES NOT** work if no user is logged in (an error status will be returned).
- The **retrieve** command will read a previously uploaded submission by the user and download it to the specified path.
- Example (for the same user **Alex** as in the previous example): **retrieve homework1 d:\\homework\_final.txt** will copy the contents of the file **%AppDir%\\users\\Alex\\homework1** to **d:\\homework\_final.txt**.
- The thread pool will also be used for the transfer in this case.

## 6.exit

- Performs cleanup of resources, after which the console application closes.
- This command is available at any time.

## Notes

- The application **MUST NOT crash** regardless of the input received from the user. Any crash/hang will be penalized.
- The application must **CORRECTLY manage resources**; any memory leak will be penalized.
- The application must be able to create all directories and files and must function correctly between restarts (**existing users will not be deleted if the application is restarted!**).
- **DO NOT hardcode paths!**
- **It is up to you what additional measures you take to protect the application from fuzzers, brute-force attacks, and other vulnerabilities.**
- **Ensure the existing unit tests are passing before delivering your solution. This is the bare minimum for a passing grade. Submissions which do not meet this criteria will not be considered.**