**Source Code**

**File\_Input\_Driver.cpp**

#include "File\_Input\_class.cpp"

// start main

*int* main () {

    // create instance of file input class

    FileInput file\_obj;

    file\_obj.get\_input();

    file\_obj.to\_file();

    file\_obj.reverse();

    file\_obj.to\_reverse();

    return 0;

} //end main

**File\_Input\_Class.cpp**

/\*

User Input Program

CSC 450 Programming III

Johnny Olmedo

\*/

#include <iostream>

#include <iomanip>

#include <ostream>

#include <fstream>

using *namespace* std;

*class* FileInput

{

*public:*

*void* get\_input();

        // this function will get input from user

*void* to\_file();

        // this function will write to file

*void* reverse();

        // this function will reverse user input

*void* to\_reverse();

        // this function will write to file again

*private:*

        string user\_input;

};

*void* FileInput::get\_input(){

    cout << "Please Enter any input: ";

    getline(cin, user\_input);

}

*void* FileInput::to\_file() {

    fstream toFile;

    try {

        toFile.open("CSC450\_CT5\_mod5.txt", ios::app);

        if (!toFile) {

            throw "Unable to open file";

        }

        else {

            //begin writing to File.

            toFile << "\nOriginal User Input:";

            toFile << user\_input << endl;

            cout << "Data appended Succesdully \n";

            toFile.close(); //close the write stream.

        }

        cout <<"user input: " << user\_input << endl;

    }

    catch (const *char*\* msg) {

        cerr << msg << endl;

    }

}

*void* FileInput::reverse()

{

*int* n = user\_input.length();

    // Swap character starting from two

    // corners

    for (*int* i = 0; i < n / 2; i++){

        swap(user\_input[i], user\_input[n - i - 1]);

    }

    cout <<"reverse user input: " << user\_input << endl;

}

*void* FileInput::to\_reverse() {

    fstream toFile;

    try {

        toFile.open("CSC450-mod5-reverse.txt", ios::app);

        if (!toFile) {

            throw "Unable to open file";

        }

        else {

            //begin writing to File.

            toFile << "\nReverse User Input:";

            toFile << user\_input << endl;

            cout << "Reverse Data appended Succesdully \n";

            toFile.close(); //close the write stream.

        }

    }

    catch (const *char*\* msg) {

        cerr << msg << endl;

    }

}

**Screenshots:**

**Text

Description automatically generated**

**Analysis**

The C++ programming language is a powerful programming language that can manipulate computer systems and the connected files. Some of these capabilities introduce vulnerabilities in software as sometimes valuable files with critical information can be left exposed to attackers. Directory traversal vulnerabilities (also known as path traversal vulnerabilities) allow bad actors to gain access to folders that they shouldn’t have access to (Exploring, 2022). One of the main things I want to focus on while writing C++ programs is to create programs that safely write to other file systems. Let’s look at the program to see what we can find.

Main:

##include "File\_Input\_class.cpp"

// start main

*int* main () {

    // create instance of file input class

    FileInput file\_obj;

    file\_obj.get\_input();

    file\_obj.to\_file();

    file\_obj.reverse();

    file\_obj.to\_reverse();

    return 0;

} //end main

As you can see here. The main of this program is a driver file that includes the File Input class to create an instance of the object and access its data. This is a secure way of using object-oriented programming to call on different data as needed from different files. I don’t see anything wrong at this point. Next let’s look at the class to see what attributes it has.

Class:

*class* FileInput

{

*public:*

*void* get\_input();

        // this function will get input from user

*void* to\_file();

        // this function will write to file

*void* reverse();

        // this function will reverse user input

*void* to\_reverse();

        // this function will write to file again

*private:*

        string user\_input;

};

We can see here that the class has a handful of public functions that are used to manipulate data for a single private variables; user\_input. This is a secure way of writing code because this means the users data cannot be taken and used by just anyone. Next, we will look at the induvial functions.

*void* FileInput::get\_input(){

    cout << "Please Enter any input: ";

    getline(cin, user\_input);

}

We see that the get\_input function allows the user to securely input any large string.

*void* FileInput::to\_file() {

    fstream toFile;

    try {

        toFile.open("CSC450\_CT5\_mod5.txt", ios::app);

        if (!toFile) {

            throw "Unable to open file";

        }

        else {

            //begin writing to File.

            toFile << "\nOriginal User Input:";

            toFile << user\_input << endl;

            cout << "Data appended Succesdully \n";

            toFile.close(); //close the write stream.

        }

        cout <<"user input: " << user\_input << endl;

    }

    catch (const *char*\* msg) {

        cerr << msg << endl;

    }

}

Here is the classes file appending to the file function. The program implements a try, catch, throw block, which allows the program to safely exit in the case the file it’s looking for isn’t found. In the case it is found, the program will append to the file, and close it.

*void* FileInput::reverse()

{

*int* n = user\_input.length();

    // Swap character starting from two

    // corners

    for (*int* i = 0; i < n / 2; i++){

        swap(user\_input[i], user\_input[n - i - 1]);

    }

    cout <<"reverse user input: " << user\_input << endl;

}

*void* FileInput::to\_reverse() {

    fstream toFile;

    try {

        toFile.open("CSC450-mod5-reverse.txt", ios::app);

        if (!toFile) {

            throw "Unable to open file";

        }

        else {

            //begin writing to File.

            toFile << "\nReverse User Input:";

            toFile << user\_input << endl;

            cout << "Reverse Data appended Succesdully \n";

            toFile.close(); //close the write stream.

        }

    }

    catch (const *char*\* msg) {

        cerr << msg << endl;

    }

}

In the two functions above, we have the reverse and reverse to functions. These functions reverse the given input and write them back to the reverse file specified in the assignment. I will admit I was having issues getting the program to catch if the file didn’t exist, as of right now it just created a file if it doesn’t exist, this a security vulnerability that needs to be fixed. Getting the try catch block to work is the key.

Finally, I believe this code looks secure for the most part. The only thing I would add to improve functionality and security of the code are working try, throw, and catch blocks. Also, I would implement the use of printf functions for display. Printf can either be a self-contained string to be printed, copied, and it can draw from other variables to build the final string. A failure to use this format string properly creates vulnerabilities in a C++ application (Howard, 2021).

**References**

Exploring 3 types of directory traversal vulnerabilities in C/C++. (2022, April 4). Snyk. <https://snyk.io/blog/exploring-3-types-of-directory-traversal-vulnerabilities-in-c-c/>

Howard Poston. (2021, November 3). Format string vulnerabilities. Infosec Resources. <https://resources.infosecinstitute.com/topic/format-string-vulnerabilities/>