**Security Principles and Validation of Security Principles – CS2S562 – Part 1**

Before creating the contents of my application, I initially implemented a Secure Logger into a blank Visual Studio project with the help of the UML class diagram shown below.

Logger

+virtual void log ()

<<interface>>

LoggerDecorator

-contents

+log ()

Main

<<client>>

HTMLLogger

+makeHTML ()

+log ()

EncryptLogger

-encryptDecrypt ()

+log ()

FormatFactoryM

+formatLogger (logger)

LoggerFactoryM

+getLogger ()

ConsoleLogger

+log ()

TxtFileLogger

+log ()

BinFileLogger

+log ()

*creates*

*creates*

*creates*

*creates*

*creates*

*creates*

*uses*

I then implemented an Authenticator pattern to provide secure login, using the UML diagram shown below to help.

Subject

-id

+getID ()

Main

<<client>>

Authenticator

-encryptDecrypt ()

+authenticateUser ()

AuthenticationInfo

-map<string, string> Users

-password

-proofID

+getPassword ()

+setProofofID ()

+getProofofID ()

+validateUser (userID, password)

ProofofID

-proofID

+getProofofID ()

+setProofofID ()

checks

creates

requests

Once implementing these two design patterns, I then created the following UML class diagram to help design the contents of the application.

Main

<<client>>

Interface

+getOption ()

+processOption ()

Temperature

+setData ()

+getData ()

Electricity

+setData ()

+getData ()

+saveData ()

Internet

+setData ()

+getData ()

+saveData ()

DeviceStatus

+setData ()

+getData ()

+saveData ()

uses

uses

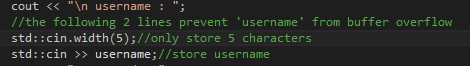
uses

uses

uses

**Character and String Vulnerabilities**

When copying data to a buffer that is not large enough to hold this data can result in a buffer overflow. Buffer overflows frequently occur when manipulating strings, therefore I have taken measures to try and avoid this. In accordance to rule five of the CERT rules, characters and strings (STR50-CPP), I have taken action to ensure declared strings have sufficient storage for character data and the null pointer. Below are two lines of code showing the *width ()* method being used. This method is being used to only accept input of five characters into the username variable.



Class: Subject.cpp

Line: 9

**Integer Vulnerabilities**

In C and C++ standards, compilers are not provided with definitions on how to encode an overflow. Therefore I have taken steps to prevent this. In accordance to rule four of the CERT rules, Integers (INT32-C), I have used *size\_t* in the below for loop as I have used a size expression. In the loop, s*ize\_t* is used as it is guaranteed to be able to express the maximum size of any object.

Class: Electric.cpp

Line: 20

**Memory Management**

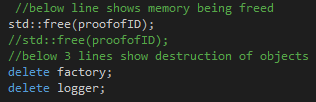
It is important to effectively construct and destruct objects when manually managing their lifetime. If not done properly, could result in loss of data or undefined behaviour. In accordance to rule six of the CERT rules, memory management (MEM53-CPP), I have ensured that all objects are effectively constructed with the sufficient memory and deallocated this memory and destructed. This is shown below where 3 objects are constructed and destructed.

Line: 14

**ALL:** Class: Main.cpp



Line: 19



Line: 42

**Input/output**

When inputting and outputting to a file, it is important that measures are taken to ensure files are properly closed, as otherwise this may enable an attacker to exhaust system resources and can increase the risk that data written into in-memory file buffers will not be flushed in the event of abnormal program termination. In accordance to rule seven of the CERT rules, Input output (FI051-CPP), I have taken measures to ensure files are closed when they are no longer needed. This is shown in the image below.

