# Application Security Strategy

# Jake Landers

# CI 330: Professor Reid - Embry Riddle Aeronautical University

# Table of Contents:

Cover Page.....................................................................................1

Introduction....................................................................................3

SDLC..............................................................................................3-7

SSDLC............................................................................................3-21

i. Security Requirements Gathering.............................................8-14

ii. Threat Modeling......................................................................14-16

iii. Coding Standards....................................................................16-20

Use and Abuse Cases.......................................................................21

Functional Requirements.................................................................22-23

Metrics.............................................................................................23-25

Third Party Testing..........................................................................26-32

Conclusion.......................................................................................33

References........................................................................................34

# Application Security Strategy

The application for either sensor must be secured. Sensors on a pipeline may not seem like something that would get attacked, but hackers will invade anything they can if there is a service they can disrupt, or information they can steal. To develop my application security strategy, I will start by developing an SDLC and a SSDLC. Within the SSDLC, I will discuss the threat model, coding standards, and security requirements gathering. Then, I will state the use cases and state an abuse case for every use case. The use cases will be used to help develop functional requirements. Finally, I will discuss metrics, 3rd party tools I could use to aid in security for my application, and I will describe how my code works.

**The Software Development Lifecycle:**

The Software Development Lifecycle, or SDLC for short, is used to provide an understanding of software and the process of building that software. The goal of building an SDLC is to improve the development process and the quality of the software overall. The phases of the SDLC are planning, defining, designing, building, testing, and deployment. My SDLC is described below:

Planning:

1. Devices will have solar power batteries and Wi-Fi capabilities.
2. Design a log in page for username and password credentials.
3. Give users the option to reset their password if they need to.
4. Give each sensor an IP address.
5. Program each sensor to display the route of the crude oil when prompted: Routes will be either Spokane, WA or Great Falls, MT.
6. Program sensors to display flow level: Pipe is 48 inches in diameter.
7. Program sensor to display flow speed: Want to maintain flow speed of 7 mph.
8. Program sensor to display viscosity: want 500 centistokes at 40 degrees Celsius.
9. Program sensor to display temperature inside and out.

* Inside: Want temperature of 40 deg C.
* Outside: 110-degree temperature outside of the pipeline could be a disaster.

1. Program sensor to display GPS coordinates.

* Change in distance between pipeline can identify tension.
* Display warning if change in distance is detected.

1. Program sensor to display battery level.
2. Program sensor to display available hard drive space.
3. Program sensor to display Wi-Fi strength.

* Signal strength reported every 30 seconds.
* Device stores data if Wi-Fi is lost until it is back again.
* Data is erased from sensor once it is reported.
* Sensors authenticate into database before sending data.
* Default username and password are both VT235x-c.
* Default user can be changed, which user device belongs to must be kept track of.
* One device every 3 miles.

Defining:

1. Give each sensor an IP address:

* Number of IP addresses needed will be equal to number of devices needed.
* Number of devices needed can be calculated by: Number of miles of pipeline/3 since the sensors are to be put every 3 miles.

1. Display route at each sensor:

* Options are Spokane or Great Falls.

1. Program sensor to display flow level:

* Flow level will be displayed when the user logs in and chooses a sensor by IP address.
* If flow level is above or below what it should be, give the user a warning.

1. Program sensor to display flow speed:

* If flow speed is not what it should be, give the user a warning.

1. Program sensor to display viscosity:

* If viscosity is not what it should be, display a warning.

1. Program sensor to display temperature inside and out

* If the temperature is not in the correct range, give a warning.

1. Program sensor to display GPS location.

* If GPS location changes, give a warning.

1. Program sensor to display battery level.

* If battery level drops below certain percentage, or it is losing battery every day, give a warning.

1. Program sensor to display available hard drive space

* If hard drive space is increasing instead of emptying every 30 seconds, give a warning.

1. Program sensor to display Wi Fi strength:

* Signal strength reported every 30 seconds.
* If signal strength isn't reported every 30 seconds, give a warning.
* If signal strength is low, give a warning.

1. Erase data from sensor after it is reported to database

* Make sure sensor does not erase data until database confirms that it has stored the data that the sensor sent to it.
* Sensor must authenticate into database before it can send data

Designing:

Log in page:

* Include text boxes for username and password input.
* Add input validation to the code.
* Add a button the user can click to reset their password.

Sensor Page

* Add button that allows user to choose sensor by IP address.
* Once the user selects an IP address, return the IP address to the previous page and display it on that page.
* After choosing a sensor, display the flow route, flow level, viscosity, temperature, flow speed, and horizontal and vertical distance between sensors.
* Add a button that will allow the user to generate a file of database data but require the admin log in credentials.
* Add a button that displays the battery life, free hard drive space, GPS location, and signal strength of the sensor.
* Add a button that lets you choose a second sensor and display the distance between the first sensor you chose and the new sensor.
* Add a button that allows the user to configure a sensor but require the admin credentials.

On IP Addresses Page:

* Add a button to generate the list of IP addresses.
* Display a list of the IP addresses and allow the user to click on one.
* The IP address they select will appear in the text box.
* Add a button that allows the user to confirm their decision and return to the previous page.

On Device Specs page:

* Add a button that displays the data.
* Display battery life, hard drive space, signal strength, and GPS location.

The building, testing, and deployment phases were done in my Visual Studio development environment.

**Software Security Development Life Cycle:**

The phases of the Software Security Development Lifecycle are:

1. Requirement

2. Design

3. Development

4. Testing

5. Deployment

6. Maintenance

The Software Security Development Life Cycle, or SSDLC, will consist of security requirements, secure coding standards, a threat model, secure coding practices, penetration testing and finally secure deployment. First, I will use my list of use and abuse cases to come up with security requirements for the requirement phase:

**Security Requirements Gathering:**

The use case of logging into the device can be abused by hackers in several different ways. What makes this use case easy to abuse is that the username and password for the device are both the name of the device. These credentials are so simple that a hacker could guess them. A security requirement that will fix this abuse case is requiring a log in with authorized user credentials to access the log in page for the sensor where the default username and password is the name of the sensor. The username and password of the sensor will still be insecure, but it will be protected from unauthorized access by requiring log in credentials before accessing the log in page for the sensor.

Here is an example of code that can be used to validate whether an authorized user credentials were entered:

**(Security Requirements Gathering PowerPoint)**

if (textBox1.Text != usernameVT && textBox2.Text != passwordVT)

MessageBox.Show("Unsuccessful log in attempt. Please try again.");

This if statement will check if the username and corresponding password are valid. This will prevent the hacker from entering random credentials into the user interface and logging in to the application.

The use case of entering log in credentials can be abused with SQL injections. SQL injections can happen at any log in page of this app, so the log in pages should be configured to check for SQL queries. The if statement above isn't enough to protect from SQL queries being entered as the username and password. The following is an if statement and function that can be used to check for SQL queries at the log in page:

static bool findWord(String text, String word)

{

String[] s = text.Split(' '); //text variable is user input for log in page, words are split by spaces (' ')

foreach(String temp in s) //Each word passed with text variable is saved in temp variable

{

if (temp.CompareTo(word) == 0)

return true; //return true if the word is the same

}

return false; //return false if the word is not the same

}

string sql1 = "SELECT";

string sql2 = "CHOOSE";

string sql3 = "WHERE";

string sql4 = "COUNT";

string sql5 = "=";

string sql6 = "\*";

string sql7 = "WHERE";

string sql8 = "AND";

if (findWord(textBox2.Text, sql1) == true || findWord(textBox2.Text, sql2) == true ||

findWord(textBox2.Text, sql3) == true || findWord(textBox2.Text, sql4) == true ||

findWord(textBox2.Text, sql5) == true || findWord(textBox2.Text, sql6) == true ||

findWord(textBox2.Text, sql7) == true || findWord(textBox2.Text, sql8) == true ||

textBox2.Text != username)

{

MessageBox.Show("Unsuccessful log in attempt. Please try again.");

textBox2.Clear();

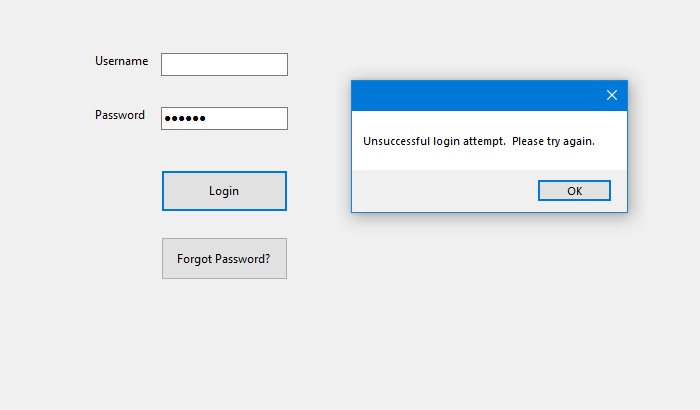
}

**(https://www.geeksforgeeks.org/c-sharp-string-contains-method/)**

**(Input Validation in Web Applications to Prevent SQL Injection PowerPoint)**

Common SQL queries are saved as variables, and those variables are all compared with what the user enters in the log in text box. If any of those variables show up as a log in credential, it will throw an error and prompt the user to try again. This can also be implemented for the password text box. Checking the log in page for SQL queries will be a security requirement.

Vague error messages will also be a security requirement. Part of input validation is error messages. Error messages should be vague rather than specific. If an error message is too specific it can give a hacker information that helps them bypass the security of the log in page. For example, if "Incorrect Password" is given as an error message, the hacker can use this to confirm that they at least know a valid username. Here is an example of an error message that doesn't give away any useful information:

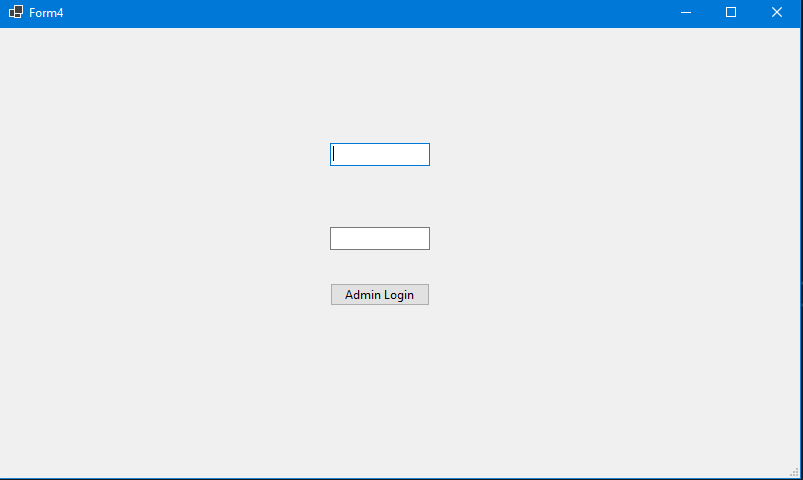
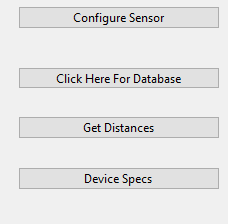


**(Secure Coding Practices for Error Handling PowerPoint)**

The application must allow the user to select a sensor and display the sensors information once the user has successfully logged into the app. This information includes the device name, IP address, pipe route, flow level, flow speed, viscosity, temperature, and hard drive space. This will seem obvious, but the app should be configured so that this information is displayed only when someone successfully logs in to the app and must be a security requirement.

The sensors are configured to send data to the database every 30 seconds, and it should wipe the data from its own hard drive once it is reported to the database. If the sensor fails to erase data after sending it to the database, the data will pile up on the sensor. This could pose a problem to the pipeline because if the hard drive on the sensor is full and something is wrong with the pipeline, the sensor won't be able to report a warning to the database so someone can respond to it. This is an example of when a functional requirement and a security requirement can be the same thing.

If the user wants to change the functionality of the sensor in anyway, they must have access to the admin credentials, and only a person in power such as a manager will have these credentials. This will add security because if a hacker gets ahold of someone's log in credentials, they will also have to locate the admin credentials if they want to alter how the sensor works. Here is an example of how the code works:



private void button3\_Click(object sender, EventArgs e)

{

Form4 f4 = new Form4();

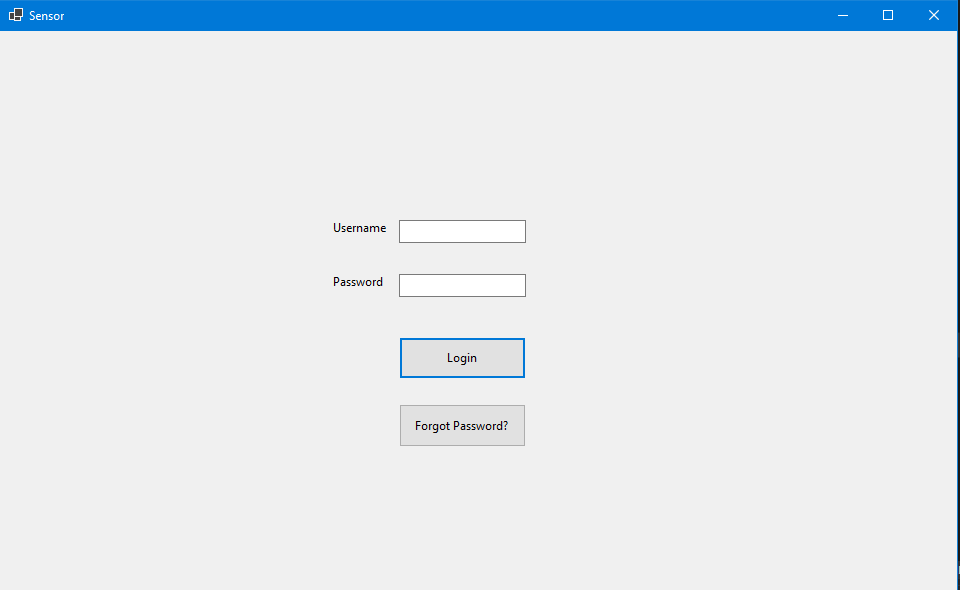
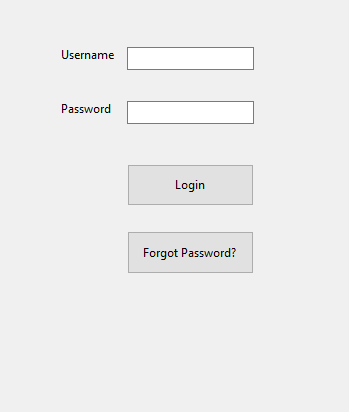
f4.ShowDialog();

}

(https://www.youtube.com/watch?v=mjOp4jCTMGI)

button3\_Click is the code segment responsible for the "Configure Sensor" button. Any code that you put within that function will be executed when you click the "Configure Sensor" button. All I want to happen when that button is clicked is for the prompt for admin log in to appear, which is what f4.ShowDialog(); does.

I would also require the admin log in page to appear anytime someone wants to change their password to prevent an attacker from changing it remotely. This would require someone to meet with a person in charge and have them enter the admin credentials to allow the password change. This way there is no denying that it is indeed an authorized user that wants to change their password. Here is an example of how this could be implemented:



private void button2\_Click(object sender, EventArgs e);

{

Form4 f4 = new Form4();

f4.ShowDialog();

}

button\_Click2 is the function for the "Forgot Password?" button. Any code put within this function will execute when the "Forgot Password?" button is clicked. All we must do is put code in the function that brings up the admin log in page. The admin log in page is known as Form 4, so all the code must do is instantiate Form4 in the button click event, and display Form 4 with f4.ShowDialog(); when the button is pressed.

The app has a button that allows the user to take data from the database and write it to a .csv file on the user's device so the user can view data from the database. This should be secured with another prompt from the admin log in page so only people in charge can have access to this sensitive data should they need it.

The log in pages should only allow a certain number of log in attempts. This will prevent an attacker from implementing a brute force log in attempt on the log in page. The user should be given no more than 5 attempts to log in to the application. This will give the user plenty of chances to successfully log in if they make a mistake the first time or two that they try to log in. The user should be allowed to log in again once the admin credentials have been entered by a person in charge. Here is an example of an if statement that can be used to check if someone has failed to log in too many times:

if (findWord(textBox2.Text, sql1) == true || findWord(textBox2.Text, sql2) == true ||

findWord(textBox2.Text, sql3) == true || findWord(textBox2.Text, sql4) == true ||

findWord(textBox2.Text, sql5) == true || findWord(textBox2.Text, sql6) == true ||

findWord(textBox2.Text, sql7) == true || findWord(textBox2.Text, sql8) == true ||

textBox2.Text != username)

{

MessageBox.Show("Unsuccessful login attempt. Please try again.");

textBox1.Clear();

textBox2.Clear();

loginAttempts += 1;

MessageBox.Show(loginAttempts.ToString());

if (loginAttempts == 5)

{

MessageBox.Show("Too many login attempts.");

Form4 f4 = new Form4();

f4.ShowDialog();

this.Hide();

loginAttempts = 0;

}

The first if statement is responsible for checking if the user entered an SQL query, or if the user entered an invalid password. If either of those criteria are met, the loginAttempts variable will increase by 1. The second if statement is entered if the loginAttempts variable reaches a value of 5. Once that happens, the admin log in page opens and the admin credentials must be entered if the user wants to keep trying to enter their credentials again.

**(Secure Coding Practices for Input Validation - CASE-Net.pdf)**

Security Requirements Gathered:

1. Require authorized user log in page to access log in page for sensors.

2. Check log in page for SQL queries.

3. Vague error messages.

4. Don't display sensor information until valid credentials are entered.

5. Erase data from hard drive after reporting to database.

6. Require admin log in to change functionality of sensor.

7. Require admin log in to change password.

8. Require admin log in to view CSV file from database.

9. Only allow limited number of log in attempts.

The design phase is next. The design phase incorporates the security requirements listed above but will also include secure coding standards and the threat model.

**Threat Modeling:**

Threat modeling is helpful because it allows you to address security issues early in the project which will prevent you from writing insecure code in the first place. Threat modeling also helps you prioritize threats based on severity. For this application, the DREAD threat model should suffice.

**(Application Security Threat Modeling.pdf, Page 13)**

DREAD stands for:

1. Damage Potential

* How much damage can a threat cause?

2. Reproducible

* Can the threat be used in other parts of the app?

3. Exploitability

* What would someone need to exploit the threat?

4. Affected Users

* Who are the victims of the threat being exploited?

5. Discoverability

* Is the threat easy to find?

This application only does two things:

1. Takes user input.

2. Display's information.

The only real threat in my application lies with the user input. This is where an attacker can inject malicious code or enter leaked employee log in credentials and gain access to the application. There are several prompts for user input with this application:

1. The first log in page.

2. The admin log in page that appears if the "Forgot Password" button is pressed.

3. The admin log in page that appears if the "Configure Sensor" button is pressed.

4. The admin log in page that appears if the "Database" button is pressed

5. The log in page for the device that was chosen.

I will incorporate the log in pages into the DREAD model and describe why they belong there.

Damage Potential:

* The log in page has the highest risk associated with it because if a hacker cannot get passed the log in page there is nothing they can do.
* The "Forgot Password" page that appears when you click the button has the next highest risk. This would allow a hacker to change a user's credentials, potentially the credentials of an admin user, and lock them out of the system while the attacker wreaks havoc.
* The "Configure Sensor" page that appears when you click the button has the next highest risk. This would allow the attacker to disrupt the pipeline if they got access to it.
* The "Database" button poses the next highest threat. This wouldn't cause any damage to the system, but it would compromise the confidentiality of the data in the database.
* The log in page for the chosen device has the lowest risk. This page is only vulnerable to SQL injection attacks because if an attacker can get to this page that means they have already hacked the log in page. This log in page and the ones above are vulnerable to SQL injection.

Reproducible:

* All of the threats are reproducible. Any of the log in pages are vulnerable to stolen credentials or SQL queries being entered into them.

Exploitability: What would someone need to exploit the threat?

* Someone would need to obtain valid log in credentials or execute the right SQL query at the log in page to exploit the threat.

Affected Users:

* Everybody could be affected by an attack at the log in page. If the credentials of the authorized and admin users were stolen a hacker could use these to alter devices on the pipeline. If an attacker can execute an SQL query in the username prompt, it can cause damage to the data in the database.

Discoverability: How easy is this threat to discover?

* These threats are easy to discover because there is already plenty of information about how log in pages can be abused.
* **(Application Security Threat Modeling.pdf, Page 37)**

**Coding Standards:**

My coding standards for this application will emphasize safe/secure coding practices and the readability of code. The readability of code involves indentation, where to put brackets, adding comments, and incorporating variable and function names that are descriptive of what each is meant for. I will start with readability:

Code should be as easy to read as possible to allow other programmers or developers to understand the code even if they weren't involved in writing it, to make third-party audits more efficient, and so the programmer who wrote the code doesn't struggle to understand what they did if they haven't seen the code in a while. Code should be written in an organized manner, which means you should use a consistent indent size throughout the program and brackets should be in the same place for all functions. In my application, brackets make up the first and last lines of a function. Variable names and function names should be as descriptive as possible so the person reading the code knows what it is meant for just by reading it. For example, you should never name a variable "i" unless it is an iterator for a loop. If you have a string variable that stores the name of something, the name of that variable should be the name of that thing. The same applies to naming functions. For example, if you write a function that returns the square root of a number, you could name the function "squareRootFunction".

Comments are crucial for helping people understand code. Writing organized code and using good variable and function names can tell a person a lot about a program, but some algorithms are more complex than others and comments can be necessary to help someone understand what the code does. Comments should be as descriptive as possible, and comments should either be placed above the line of code that the comment explains or right next to it. If the comment is going to take up a lot of space, the comment should be placed above the line of code it explains rather than next to it.

The safe/secure coding practices for this application should be centered around input validation. Prompts for user input are a big attack vector for hackers, therefore they need a lot of attention. The user input for log in credentials must be protected against invalid credentials and most importantly SQL injection attempts. Invalid credentials can be protected against with while loops or if statements where the conditions are that the values entered for username and password must both be usernames and passwords that exist, and they must be the username and password combination that correspond to each other. Here is a coded example of how this can be implemented:

while (textBox2.Text != usernameVT && textBox1.Text != passwordVT)

MessageBox.Show("Unsuccessful log in attempt. Please try again.");

The while loop will check if the user entered a valid username and if the password they entered corresponds to that username. If either of the two are wrong, it will continue to prompt the user until they get it correct. A variable that keeps track of the number of log in attempts should be implemented, and if a certain number of log in attempts are exceeded an error should be given and the user shouldn't be allowed to log in again until the admin credentials are entered.

if (loginAttempts == 5)

{

MessageBox.Show("Too many login attempts.");

Form4 f4 = new Form4();

f4.ShowDialog();

this.Hide();

loginAttempts = 0;

}

**(https://www.perforce.com/resources/qac/coding-standards)**

**(CERT C++ Coding Standard 2016.pdf)**

The development phase of the SSDLC is next. The development phase incorporates the coding standards from the design phase and secure coding practices. For this phase I will demonstrate how my coding standards can be implemented.

The biggest threats to this app are the log in pages. The log in pages allow an attacker to pass SQL queries to the database to steal information or disrupt the service. The log in page must also be configured so a user must enter username and passwords that exist and are correct to prevent an attacker from bypassing the log in page with random credentials. Here is an example of how SQL injection can be prevented:

static bool findWord(String text, String word)

{

String[] s = text.Split(' '); //text variable is user input for log in page, words are split by spaces (' ')

foreach (String temp in s) //Each word passed with text variable is saved in temp variable

{

if (temp.CompareTo(word) == 0)

return true; //return true if the word is the same

}

return false; //return false if the word is not the same

}

string sql1 = "SELECT";

string sql2 = "CHOOSE";

string sql3 = "WHERE";

string sql4 = "COUNT";

string sql5 = "=";

string sql6 = "\*";

string sql7 = "WHERE";

string sql8 = "AND";

if (findWord(textBox2.Text, sql1) == true || findWord(textBox2.Text, sql2) == true ||

findWord(textBox2.Text, sql3) == true || findWord(textBox2.Text, sql4) == true ||

findWord(textBox2.Text, sql5) == true || findWord(textBox2.Text, sql6) == true ||

findWord(textBox2.Text, sql7) == true || findWord(textBox2.Text, sql8) == true ||

textBox2.Text != username)

{

MessageBox.Show("Unsuccessful log in attempt. Please try again.");

textBox2.Clear();

}

This if statement will use the findWord function to compare what the user entered in the log in page to see if they entered any SQL queries. If an SQL query is found, the user input is cleared, and the user will be prompted to try again. This will repeat until the user does not enter any SQL queries as log in credentials.

Here is how I can implement my secure coding standards to make sure a correct username and password is entered:

string usernameVT = "username";

string passwordVT = "password";

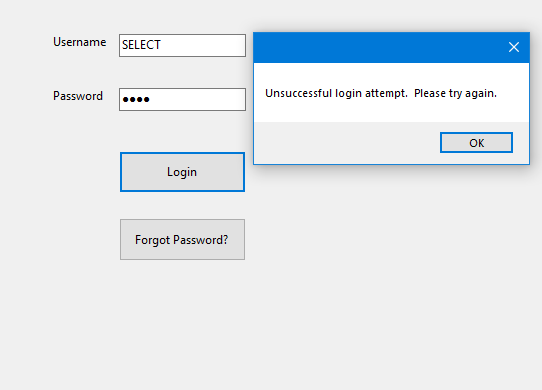
while (textBox1.Text != usernameVT && textBox2.Text != passwordVT)

MessageBox.Show("Unsuccessful log in attempt. Please try again.");

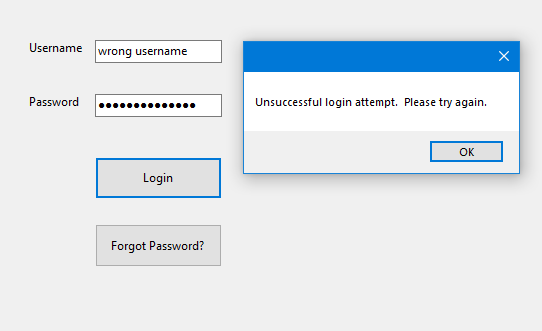
The username and password are not secure, but this is just an example of how the input validation will be configured to make sure the correct username and password are entered. The while loop will check the username and password textbox in the app and compare the values to the values that are known to be a username and its corresponding password. If the two don't match, an error message will be displayed, and the user will be stuck in that while loop until they enter the correct credentials.

The testing phase must be implemented in my development environment, but here are a few images that show my testing was successful:

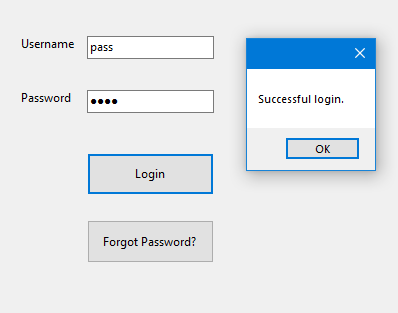
Checking for an SQL query:



Incorrect username and password:



Valid username and password:



The deployment and maintenance phases will be done in my Visual Studio development environment.

Use and abuse cases should be gathered before coding begins. It should be done this way because this allows you to have a list of things that the code should and should not do, and you can keep a list of use and abuse cases handy while you are coding so you can keep track of what has and has not been done. The following are use and abuse cases for the VTX235x-c:

|  |  |
| --- | --- |
| Use Case | Abuse Case |
| Connect to Wi-Fi | Attack Vector for Hackers |
| Default Log in Credentials are Sensor Name | Too Easy to Guess Log in Credentials |
| Enter Log in Credentials | SQL Injection |
| Display Sensor ID | Target a Specific Sensor |
| Display Sensor IP | Ping/Connect to a Specific Sensor |
| Display Pipe Route | Target a Specific Pipe Route |
| Display Flow Level | Determine What is a Bad Flow Level |
| Display Flow Speed | Determine What is a Bad Flow Speed |
| Display Viscosity | Determine What is a Bad Viscosity |
| Display Temperature | Determine What is a Bad Temperature |
| Display Hard Drive Space | How Much Data to Overflow Hard Drive |
| Erase Data After It's Sent to Database | Hacker Can Steal Data if Not Erased |
| Store Data in Drive if Wi-Fi is Lost | Hack Device to Delete Data Instead of Store it |
| Default Credentials are Sensor Name | Too Easy to Guess Credentials |
| Default User Can Be Changed | Hacker Can Make Themselves Default User |

**(Security Requirements Gathering PowerPoint)**

**Functional Requirements Gathering:**

Defining use cases makes it easier to define functional requirements. Use cases and functional requirements can be described as "What is the code supposed to do?" A lot of the application has already been described, so I will use the information above plus the use and abuse cases to develop my list of functional requirements:

1. Allow user input for a username and a password.

* This is the first thing the user will see when they open the application. There will be two text boxes, one for a username and one for a password.

2. There must be a button that the user can click to submit the log in credentials that they entered. If the credentials are correct, the user should be taken away from the log in page and to the next page.

3. Include a button the user may click if they forgot their password. This button should take the user to an admin log in page where a manager can enter their credentials to authorize the password change.

4. The page that shows up after the user enters valid credentials will be the page where they get to select a sensor. A list of IP addresses will be displayed and the user must be allowed to click one of the IP addresses. Once the user clicks an IP address, the corresponding device name will appear, and there should be a button that the user can click to log in to the device from there.

5. After the user logs into the device, the page they are on will close and the next page that displays sensor information should appear. There will be a brief pause, but then the sensor information will be displayed to the screen.

6. From this page, the user should have several options:

* Display device specifications (battery life, hard drive space, signal strength, etc.)
* Configure the sensor
* Get a csv file of information from the database
* Get the distance between two sensors
* Choose a different device

These options should be available to the user as buttons they can click. The configure sensor and get a csv file from the database buttons should require authentication via the admin credentials before access to those pages is granted.

**(https://www.youtube.com/watch?v=i-QyW8D3ei0)**

**Metrics:**

Quote from Lord Kelvin: "If you can't measure it, you can't improve it". A metric is a system of measures that allow you to quantify some characteristic. Metrics are used in software development and software security to measure specific things about the project so you can improve upon them.

**(App Sec Metrics.pdf, Page 4-5)**

Here is an example of what metrics are important to keep track of when developing an application:

1. Lines of code

2. Language(s) used

3. Imports, modules, and libraries used.

4. How is sensitive information protected?

5. What vulnerabilities have been identified?

6. Is there documentation for the application?

7. Is the documentation accurate and/or complete?

8. Who were the developers?

9. What assurance activities were performed?

10. Did the assurance activities have an outcome? If so, what were they?

Numbers 1 through 5 are called direct metrics and numbers 6 through 10 are called indirect metrics. The difference between the two is that direct metrics are often things that can be counted directly, like the number of people in a room or the number of cars in a parking lot. Indirect metrics often involve yes or no questions, or questions like "who did this?" or "what did they do?" opposed to questions like "how many people did this?" or "how much did they do?". The metrics that need to be counted for this application are as follows:

Direct metrics:

1. Number of lines of code.

* This metric is easy to obtain because most development environments count the lines of code for you. This is an important metric because the number of lines of code can provide you with an estimate for how many vulnerabilities could be present, or how much work it will take to find and fix vulnerabilities and other flaws.
* My project has about 650 lines of code. This may seem like a lot, but C Sharp code often has a lot of lines that are empty, but it still counts them as lines.

2. Number of prompts for user input.

* A prompt for user input in the context of my application is any space that the user can click on and type something into and submit it. It is important to count these and count the number of them that have been secured so you know for sure if it has been done or not.

3. Languages used:

* For my application, only C Sharp was used to write the code. This metric is important because when you want to hire someone to do a third-party audit of your code, the person or company you reach out to must have personnel that are able to read the programming language that you used to develop your application.

4. Number of libraries used.

* Libraries are brought into the development environment by the user as needed. When a library is imported into a programming environment, it greatly expands the capabilities of that programming environment by providing functions and methods that were not previously there. For example, in Python you cannot graph anything until you import the "MatPlotLib" library. This metric is important to keep track of because knowing what libraries the code is using will tell you exactly what the program can do. For example, if you are in a Python environment and you see that MatPlotLib is not imported, you know for a fact that the program does not involve graphing. The only imports my application uses are:
* using System;
* using System.Windows.Forms;

5. Number of forms

* A form in C Sharp is a window that the user can interact with. My application has five forms.

**(App Sec Metrics.pdf, Page 8-10)**

**(App Sec Metrics.pdf)**

**Third Party Assurance:**

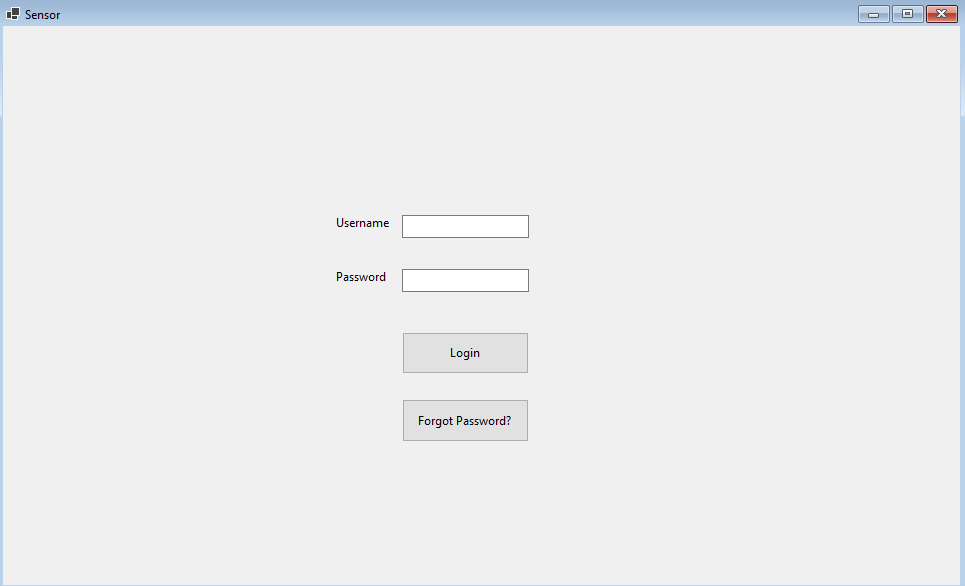
The user is considered the weakest link in the 7-domain IT infrastructure. The user is considered the weakest link because it is the only thing in the IT infrastructure that is a human. Humans make mistakes, which is why it's possible that vulnerabilities are missed when a human was doing an audit of my code. Therefore, it makes sense to use a third-party tool that can scan your application for you. This will give stakeholders peace of mind knowing that everything possible was done to make this application secure.

After doing research on the internet, I came across a product called Acunetix that I liked a lot. Acunetix can detect over 7,000 web vulnerabilities, you have the option to purchase a scan that works continuously instead of doing scans periodically, they scan for over 50,000 known network vulnerabilities, and they have multiple scan engines to speed up the process. They also provide a report on how well your application is configured to defend against the OWASP top 10 threats. They offer solutions that are tailored for a number of different industries, and the pricing is about average compared to other products. This would be overkill for the application I developed, but I guarantee if there are vulnerabilities in my code the Acunetix solution is going to find them.

https://www.acunetix.com/solutions/education/#

**Code Breakdown:**

For my application, I developed a C Sharp Windows Forms Application, or WFA for short. I went with a WFA because they are great for making user interfaces. I decided to write code for the VTX235x-c sensor. Here is how it works:



When the code is run, the user is greeted with this log in page. There is a prompt for a username and a password, a "Login" button that allows the user to submit their credentials, and a "Forgot Password?" button that allows the user to reset their password if they need it.

Here is how it works:

public Form1()

{

InitializeComponent();

}

This will cause the window for the log in page to display. The buttons and the text boxes did not require any code, C Sharp WFA's allow you to drag and drop those items wherever you need them.

Here is how I validated user input in the text boxes:

private void button1\_Click(object sender, EventArgs e)

{

string sql1 = "SELECT";

string sql2 = "CHOOSE";

string sql3 = "WHERE";

string sql4 = "COUNT";

string sql5 = "=";

string sql6 = "\*";

string sql7 = "WHERE";

string sql8 = "AND";

string username = "pass";

string password = "pass";

if (findWord(textBox2.Text, sql1) == true || findWord(textBox2.Text, sql2) == true ||

findWord(textBox2.Text, sql3) == true || findWord(textBox2.Text, sql4) == true ||

findWord(textBox2.Text, sql5) == true || findWord(textBox2.Text, sql6) == true ||

findWord(textBox2.Text, sql7) == true || findWord(textBox2.Text, sql8) == true ||

textBox2.Text != username)

{

MessageBox.Show("Unsuccessful login attempt. Please try again.");

textBox1.Clear();

textBox2.Clear();

loginAttempts += 1;

MessageBox.Show(loginAttempts.ToString());

if (loginAttempts == 5)

{

MessageBox.Show("Too many login attempts.");

Form4 f4 = new Form4();

f4.ShowDialog();

this.Hide();

loginAttempts = 0;

}

}

else if (findWord(textBox1.Text, sql1) == true || findWord(textBox1.Text, sql2) == true || findWord(textBox1.Text, sql3) == true || findWord(textBox1.Text, sql4) == true ||

findWord(textBox1.Text, sql5) == true || findWord(textBox1.Text, sql6) == true ||

findWord(textBox1.Text, sql7) == true || findWord(textBox1.Text, sql8) == true ||

textBox1.Text != password)

{

MessageBox.Show("Unsuccessful login attempt. Please try again (password).");

textBox1.Clear();

textBox2.Clear();

loginAttempts += 1;

MessageBox.Show(loginAttempts.ToString());

if (loginAttempts == 5)

{

MessageBox.Show("Too many login attempts.");

Form4 f4 = new Form4();

f4.ShowDialog();

this.Hide();

loginAttempts = 0;

}

}

else if (textBox2.Text == username && textBox1.Text == password)

{

MessageBox.Show("Successful login.");

this.Hide();

Form3 f3 = new Form3();

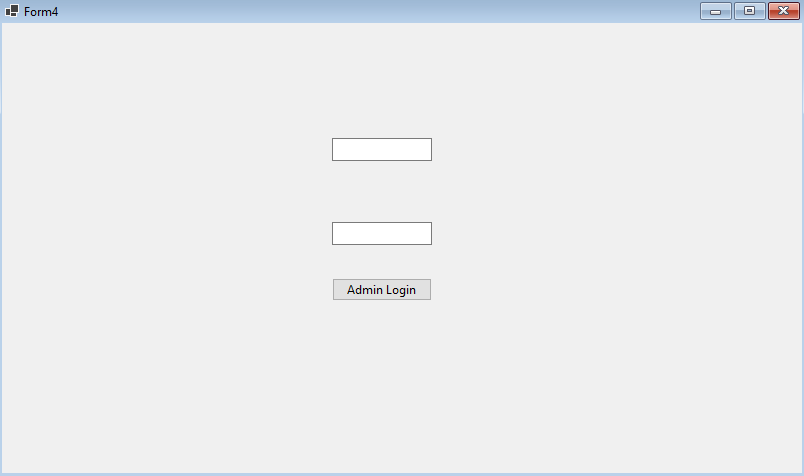
f3.ShowDialog(); //Bring the user to form 3 if the credentials are correct

}

}

I put the code within the button1\_Click() function because button1 is related to the Login button, and that is when I want the code that validates input to run. The code works by saving some common SQL queries as variables, and using the function called findWord() to compare the word the user entered in the username or password box with each of the SQL variables. If the user input matches any of the SQL queries, it will display an error message and tell the user to try again. The if statement that checks for SQL queries also compares the user input with the known username and password. If the username and password information are incorrect, an error message will be displayed, and the user will be asked to try again.

I created a variable called loginAttempts. I created this variable to keep track of the number of times a user fails to log in. If the user fails to log in 5 times, an error message is displayed, and a window appears that requires the admin login credentials to be entered before the user can try entering their credentials again:

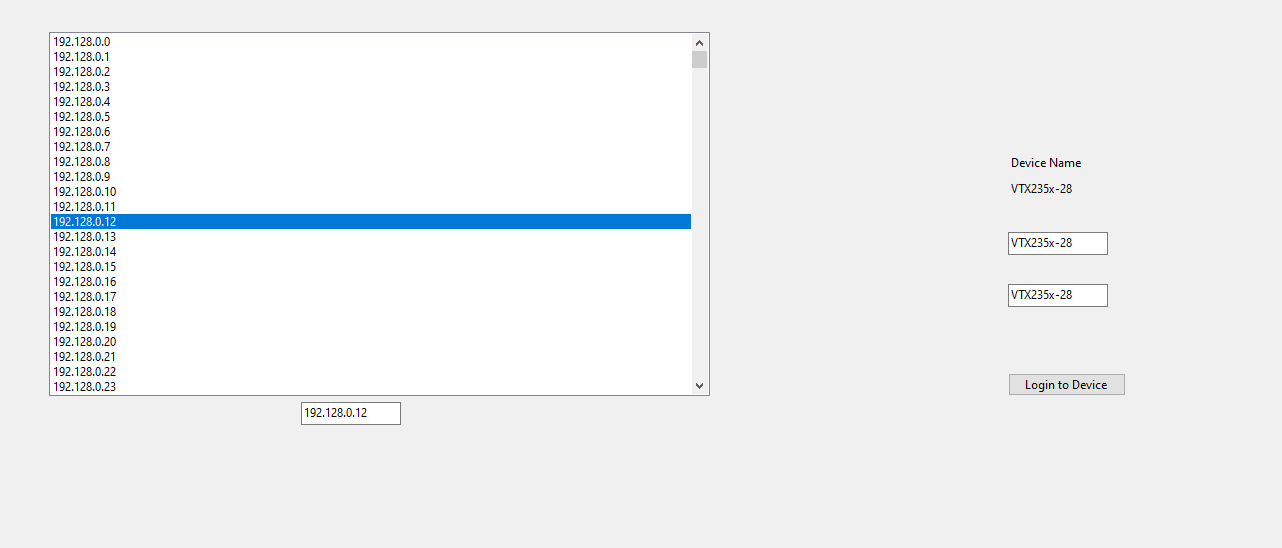


Once the user succeeds at logging in, I used this code to bring the user to the next page:

Form3 f3 = new Form3();

f3.ShowDialog();

The page the user is brought to is the page that lets them choose a sensor. A list of IP addresses is displayed, and the user can click on the one that they want.



Since the credentials for the sensor are the name of the sensor, this information is automatically put into the text boxes for the username and password, and all the user must do from there is click "Login to Device". Of course, none of these IP addresses are real, so here is the code I used to generate and display them:

string ipAddress1 = "192.128.0.";

int x = 0;

while (x <= 255)

{

string newIP = ipAddress1 + x;

listBox1.Items.Add(newIP);

x += 1;

}

string ipAddress2 = "192.128.1.";

int y = 0;

while (y <= 255)

{

string newIP = ipAddress2 + y;

listBox1.Items.Add(newIP);

y += 1;

}

string ipAddress3 = "192.128.2.";

int z = 0;

while (z <= 255)

{

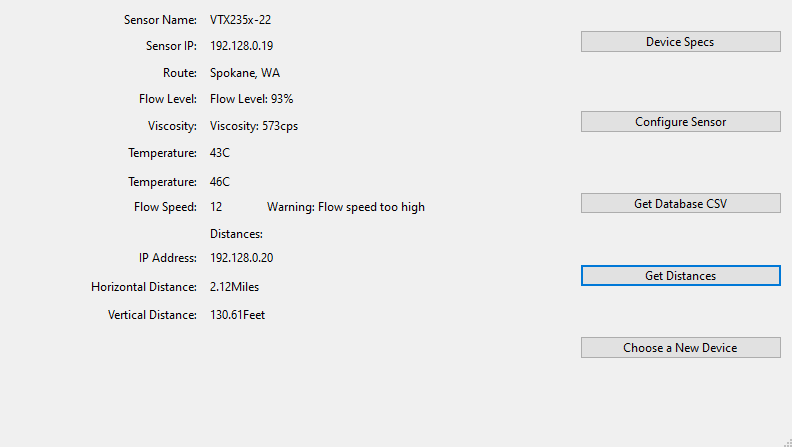
string newIP = ipAddress3 + z;

listBox1.Items.Add(newIP);

z += 1;

}

Once the user chooses an IP address and logs into the device, it brings them to the next form:



Since this is not a real sensor, most of these values are randomly generated using simple random number functions like this:

private readonly Random randomValue = new Random();

private int value(int min, int max)

{

return randomValue.Next(min, max);

}

If the user clicks the Get Database CSV button, a csv file of information from the VTX235x-c will be generated onto the user's device. Here is the code I used to do this:

string databaseFile = @"C:\Users\lande\source\repos\Sensor Application\Sensor Application\database.csv";

using (System.IO.StreamWriter file = new System.IO.StreamWriter(@databaseFile, true))

{

for (int i = 0; i <= 1000; ++i)

{

file.WriteLine("Flow lvl." + "," + "Temp in" + "," + "Temp out" + "," +

"Visc." + "," + "Flow Spd." + "," + "Route");

file.WriteLine(flowLevel(50, 70).ToString() + "%" + "," + temperature(35, 45).ToString() + " C" + "," + temperature(0, 120).ToString() + " C" + "," + viscosity(450, 550).ToString() + " cST" + "," + speed(4, 9).ToString() + " mph" + "," + "Spokane");

}

}

Recall this is not a real sensor, so I had to generate some fake data to make it look real. This code looks like a mess, but how it works is simple:

* I used functions to generate random values to simulate a sensor that is actively reading data.
* I wrote a for loop that iterates 1,000 times, so I will have 1,000 pieces of data
* Each iteration of the for loop will generate values for flow level, temperature, viscosity, flow speed, and flow route and put them into the CSV file.
* The file path will have to be configured to the user's device for it to work, so I will include an example of the CSV file that the code generates.

The rest of the application works similar to everything I have described:

* Each form is initialized with InitializeComponent();
* I use ShowDialog() to bring the user to the next form.
* The prompts for user input are validated using the same code I showed in Form1() above.
* Since I am not pulling data from real sensors, I used while loops and a random number generator functions to make the user interface look like it is reading real data to simulate what the application would look like in the real world.

Conclusion:

For my application security strategy, I developed a Software Development Life Cycle, and a Software Security Development Life Cycle to go with it. My SSDLC incorporated a threat model, secure coding standards and practices, and security requirements. Then I discussed the use and abuse cases for the application and used these to help develop my functional requirements. Lastly, I discussed important metrics, 3rd party tools that could be use to further the security of my application, and I described how my code works. My application security strategy is not perfect, but I don't believe any strategy is. With that being said, I am confident that if my strategy were followed for an application like this it would be secure and functional.

References:

https://www.geeksforgeeks.org/check-if-a-word-is-present-in-a-sentence/

**(Security Requirements Gathering PowerPoint)**

**(**[**https://www.geeksforgeeks.org/c-sharp-string-contains-method/**](https://www.geeksforgeeks.org/c-sharp-string-contains-method/)**)**

**(Input Validation in Web Applications to Prevent SQL Injection PowerPoint)**

**(Secure Coding Practices for Error Handling PowerPoint)**

**(Secure Coding Practices for Input Validation - CASE-Net.pdf)**

**(Application Security Threat Modeling.pdf, Page 13)**

**(Application Security Threat Modeling.pdf, Page 37)**

**(**[**https://www.perforce.com/resources/qac/coding-standards**](https://www.perforce.com/resources/qac/coding-standards)**)**

**(CERT C++ Coding Standard 2016.pdf)**

**(**[**https://www.youtube.com/watch?v=i-QyW8D3ei0**](https://www.youtube.com/watch?v=i-QyW8D3ei0)**)**

**(App Sec Metrics.pdf, Page 4-5)**

**(App Sec Metrics.pdf, Page 8-10)**

**(App Sec Metrics.pdf)**

[https://www.acunetix.com/solutions/education/#](https://www.acunetix.com/solutions/education/)

<https://www.youtube.com/watch?v=mjOp4jCTMGI>

<https://www.youtube.com/watch?v=tVlYojL7iCI&t=163s>

<https://www.youtube.com/watch?v=vDpww7HsdnM&t=225s>