**Green Pace Developer: Security Policy Guide Template**



# Green Pace Secure Development Policy

## Contents

[Green Pace Secure Development Policy 0](#_Toc116263199)

[Contents 1](#_Toc116263200)

[Overview 2](#_Toc116263201)

[Purpose 2](#_Toc116263202)

[Scope 2](#_Toc116263203)

[Module Three Milestone 2](#_Toc116263204)

[Ten Core Security Principles 2](#_Toc116263205)

[C/C++ Ten Coding Standards 3](#_Toc116263206)

[Coding Standard 1 4](#_Toc116263207)

[Coding Standard 2 6](#_Toc116263208)

[Coding Standard 3 8](#_Toc116263209)

[Coding Standard 4 10](#_Toc116263210)

[Coding Standard 5 13](#_Toc116263211)

[Coding Standard 6 15](#_Toc116263212)

[Coding Standard 7 17](#_Toc116263213)

[Coding Standard 8 19](#_Toc116263214)

[Coding Standard 9 21](#_Toc116263215)

[Coding Standard 10 23](#_Toc116263216)

[Defense-in-Depth Illustration 25](#_Toc116263217)

[Automation 25](#_Toc116263218)

[Summary of Risk Assessments 26](#_Toc116263219)

[Create Policies for Encryption and Triple A 26](#_Toc116263220)

[Audit Controls and Management 28](#_Toc116263221)

[Enforcement 28](#_Toc116263222)

[Exceptions Process 28](#_Toc116263223)

[Distribution 29](#_Toc116263224)

[Policy Change Control 29](#_Toc116263225)

[Policy Version History 29](#_Toc116263226)

[Appendix A Lookups 29](#_Toc116263227)

[Approved C/C++ Language Acronyms 29](#_Toc116263228)

## Overview

Software development at Green Pace requires consistent implementation of secure principles to all developed applications. Consistent approaches and methodologies must be maintained through all policies that are uniformly defined, implemented, governed, and maintained over time.

## Purpose

This policy defines the core security principles; C/C++ coding standards; authorization, authentication, and auditing standards; and data encryption standards. This article explains the differences between policy, standards, principles, and practices (guidelines and procedure): [Understanding the Hierarchy of Principles, Policies, Standards, Procedures, and Guidelines](https://www.linkedin.com/pulse/understanding-hierarchy-principles-policies-standards-wally-beddoe/).

## Scope

This document applies to all staff that create, deploy, or support custom software at Green Pace.

## Module Three Milestone

### Ten Core Security Principles

| **Principles** | Write a short paragraph explaining each of the 10 principles of security. |
| --- | --- |
| 1. ValidateInput Data | It is essential that input from the user to make sure it is valid. Invalid input is a big reason that errors occur and could cause your system to crash. More importantly, if input isn’t validated, a malicious user could get access to information that they are not supposed to have. |
| 1. Heed Compiler Warnings | It is important to make sure that if the compiler is giving you a warning that you make sure that you take your time to see why that warning is being provided. Even if the code compiles properly, that does not mean that the code is secure and there could be threats introduced by not heeding those warnings. |
| 1. Architect and Design for Security Policies | In essence this means to plan for security policies and design your code around these policies. This also means to put security on the forefront of your design process and coding practices. |
| 1. Keep It Simple | In essence, this principle is very easy; do not write more code than you need to. By writing more lines of code than necessary, you introduce the possibility of creating more errors or vulnerabilities. |
| 1. Default Deny | The importance of denying users access by default can limit the number of vulnerabilities that are written in the code. If a user does not have access to the information, there is less they can do to cause havoc. |
| 1. Adhere to the Principle of Least Privilege | The principle of lease privilege states that every user should have access to the bare minimum so that they can still do their job. If someone has access to more information than necessary, they can (either knowingly or unknowingly) get access to information that they should not have access to. |

|  |  |
| --- | --- |
| 1. Sanitize Data Sent to Other Systems | Data that is being sent to other systems should be only what the system needs. It is possible that if you just send data over without looking it over that you could send over more than you wanted. This information could be intercepted or used by other malicious individuals and cause issues with security. |
| 1. Practice Defense in Depth | Defense in depth is the process of protecting from attacks in multiple redundant layers to make sure that attacks from malicious individuals do not infiltrate your software. Protecting your system this way, gives you a better chance of catching attacks before they get too large to manage. |
| 1. Use Effective Quality Assurance Techniques | Quality assurance techniques allow us to catch problems before they can get out of hand. By implementing proper testing techniques, we can catch problems before they become multi-account leaks. |
| 1. Adopt a Secure Coding Standard | A secure coding standard gives everyone the same template to follow and holds people to the same standard. This also puts everyone on the same security standard so there are fewer possible chances for security leaks. |

### C/C++ Ten Coding Standards

Complete the coding standards portion of the template according to the Module Three milestone requirements. In Project One, follow the instructions to add a layer of security to the existing coding standards. Please start each standard on a new page, as they may take up more than one page. The first seven coding standards are labeled by category. The last three are blank so you may choose three additional standards. Be sure to label them by category and give them a sequential number for that category. Add compliant and noncompliant sections as needed to each coding standard.

#### Coding Standard 1

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Type** | [STD-001-CPP] | If code is read into the wrong data type, the code can crash, or malicious users can get data that they should not have access to. |

| **Noncompliant Code** |
| --- |
| The code block below shows that input is read from the user, but we do not check if a number has been passed in. This can lead to issues later |
| std::cout << "Enter a number:" << std::endl;  int number;  std::cin >> number;  std::cout << "You entered: " << number << std::endl; |

| **Compliant Code** |
| --- |
| The code block below correctly makes sure that the value entered is actually a number. |
| std::cout << "Enter a number:" << std::endl;  int number;  std::cin >> number;  if (!(std::cin.good())) {  std::cout << "Invalid input: do something here" << std::endl;  }  else {  std::cout << "You entered: " << number << std::endl;  } |

| **Principles(s):** Validate Input Data: This corelates to this principle, because if data is not properly validated, there could be potential issues that lead to bigger issues including (but not limited to) a crash of the system or access to information that should not be seen by the public. |
| --- |
| Effective Quality Assurance Techniques: Without properly checking the information that is sent in from the user, we can have issues that cause bigger problems later. This is a technique that can be used as a mitigation technique to make sure that this issue does not pop up later. |
| Adopt a Secure Coding Standard: By requiring all the developers to follow the same coding standard, we can make sure that everyone accounts for the potential for data type issues. Like the last principle, this is another mitigation technique that helps us prevent future bigger issues. |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level (1-5)** |
| --- | --- | --- | --- | --- |
| High | Semi-Likely | Medium (Various more likely you can have both high and low-cost remediation for this issue) | Medium | 2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CppCheck | 2.9 | CppCheck’s Desktop Application | This application checks for errors specifically with the C++ language. This tool allows the developer to check and make sure that all of the errors are dealt with and gives a brief explanation as to why there is an error in the software. |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | [STD-002-CPP] | If you are making sure that a range is followed, and the user enters an invalid data value, they could cause havoc in the system. |

| **Noncompliant Code** |
| --- |
| The code below gets input from the user, but does not check if the value is in the proper range. |
| std::cout << "Enter a number between 1 and 5:" << std::endl;  int number;  std::cin >> number;  std::cout << "You entered: " << number << std::endl; |

| **Compliant Code** |
| --- |
| The code below asks for input from the user and then makes sure that the value is in the range. (Note: See STD-001-CPP for handling invalid data type; this block assumes that the data type is entered properly. |
| std::cout << "Enter a number between 1 and 5:" << std::endl;  int number;  std::cin >> number;  if (number > 5 || number < 1) {  std::cout << "Invalid number: do something here" << std::endl;  }  else {  std::cout << "You entered: " << number << std::endl;  } |

| **Principles(s):** Validate Input Data: This is essential to this principle because validating user input will allow us to make sure that the data entered is entered properly. If this was not followed properly, there would be issues that might lead to bigger problems in the future. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level (1-5)** |
| --- | --- | --- | --- | --- |
| High | Likely | High (If a user were to enter invalid information, they could get better | Medium-High | 2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CppCheck | 2.9 | CppCheck’s Desktop Application | This application checks for errors specifically with the C++ language. This tool allows the developer to check and make sure that all of the errors are dealt with and gives a brief explanation as to why there is an error in the software. |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | [STD-003-CPP] | It is important to make sure that a string has been entered properly and that the user’s string is correct. Invalid input can cause security vulnerabilities in the system. |

| **Noncompliant Code** |
| --- |
| The code below does not make sure that the string entered is correct and could lead to issues. |
| std::cout << "Enter a phrase:" << std::endl;  std::string phrase;  std::cin >> phrase;  std::cout << "You entered: " << phrase << std::endl; |

| **Compliant Code** |
| --- |
| This code not only check to make sure a phrase is correct, but also makes sure that the individual entered a confirmation letter properly. |
| std::cout << "Enter a phrase:" << std::endl;  std::string phrase;  std::string correct;  std::cin >> phrase;  std::cout << "You entered: " << phrase << std::endl;  std::cout << "Is this correct? (Y/N)" << std::endl;  std::cin >> correct;  if (correct.size() != 1) {  std::cout << "Invalid input: do something here" << std::endl;  }  else if (correct.compare("Y") == 0) {  std::cout <<"Thank you for verifying your entered phrase"<<std::endl;  }  else if (correct.compare("N") == 0) {  std::cout << "You did not enter that phrase" << std::endl;  }  else {  std::cout << "Invlid input: do something here" << std::endl;  } |

| **Principles(s):** Validate User Input: By not validating user input, we can introduce potential issues where the user enters invalid data. This is especially true with strings, because they can be read directly and can lead to issues that are large. |
| --- |
| Keep it Simple: If simple prompts are given directing the user to enter data, there can be a decrease in the number of issues that come up. For example, if the answer is either yes or no, the developer can ask the user for either a y or n depending on the answer. |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level (1-5)** |
| --- | --- | --- | --- | --- |
| Medium | High | Low – High (If the user enters wrong data, they could crash the system or access information that they should not have access to) | High | 2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CppCheck | 2.9 | CppCheck’s Desktop Application | This application checks for errors specifically with the C++ language. This tool allows the developer to check and make sure that all of the errors are dealt with and gives a brief explanation as to why there is an error in the software. |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | [STD-004-CPP] | If you do not make sure that the input to for and SQL inquiry, then a malicious user can gain access to information that they should not have access to. |

| **Noncompliant Code** |
| --- |
| This code block just lets the user enter whatever information and as such the user can enter 1=1 or a similar pattern to get more information than they are allowed to see. |
| std::string sql;  std::cin >> sql;  runQuery(sql); |

| **Compliant Code** |
| --- |
| This code block checks to see if there are any inputs where 1=1 or a similar pattern and shuts that search down before the information is shown. |
| //Create int pointers to compare values  int equalsPointer;  int wordLength;  int wordDifference;  //SQL inquiry string  std::string sql;  //Assign int values according to the values we are searching for  equalsPointer = sql.find("=");  wordLength = sql.find\_first\_of(" ", equalsPointer);  wordDifference = sql.find\_last\_of(" ", equalsPointer);  //Get input from the user for the search  std::cin >> sql;  //Do while loop that checks for any add ons to the end of an SQL statement; this is done by looking for '=' values in the SQL statement  do {  //If the values at the equals pointer and word difference are the same, then the rest of the values surrounding '=' needs to be checked  if (sql.at(equalsPointer + 1) == sql.at(wordDifference + 1)) {  //Changes each value by 1 to move to the next char in the values around the char '='  equalsPointer = equalsPointer + 1;  wordDifference = wordDifference + 1;  //If we have reached the '=' character than we know that both values are the same and we have a potential SQL injection attack  //This section will print out an error message to the console  if (sql.at(wordDifference + 1) == '=') {  std::cout << std::endl << "SQL Injection detected on:" << std::endl << sql << std::endl << "Search Stopped" << std::endl;  return false;  }  }  //If the values are not the same, we can just check if there are more '=' characters in the sql string  else {  equalsPointer = sql.find\_first\_of("=", (equalsPointer + 1));  wordLength = sql.find\_first\_of(" ", equalsPointer);  wordDifference = sql.find\_last\_of(" ", equalsPointer);  }  } while (equalsPointer > 0);  runQuery(sql); |

| **Principles(s):** Validate User Input: By validating that the user input is valid, we can pretty much mitigate all the potential issues with this threat. This is because, all the potential SQL injection attacks come when a user has a item = item form. We can limit this by validating that the user input is proper. |
| --- |
| Default Deny: By defaulting to denying access, the developer can make sure that the user does not have access to the information that they might want before they even have access to the information they are looking to get. |
| Adhere to the Principle of Least Privilege: This means that the user should not have access to more information than they should need. For example, if a user wanted to delete an entry, it would be beneficial to make sure that they were authorized before that permission was granted. |
| Practice Defense in Depth: This means that there are multiple lines of defense before the user can gain access to the information they are looking for. We have identified that you can limit the information that the user can gain access to by checking the strings, but this should not be the only line of defense. |
| Use Effective Quality Assurance Techniques: By using proper quality assurance techniques, we can make sure that someone who might want to get information, can not have access to that information. |
| Adopt a Secure Coding Standard: By adapting this standard, we can make sure that everyone is coding to a standard where this issues is mitigated so that it is less of a concern. |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level (1-5)** |
| --- | --- | --- | --- | --- |
| High | Low | High | Medium-High | 3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CppCheck | 2.9 | CppCheck’s Desktop Application | This application checks for errors specifically with the C++ language. This tool allows the developer to check and make sure that all of the errors are dealt with and gives a brief explanation as to why there is an error in the software. |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | [STD-005-CPP] | If memory is not protected, the user can write over data, and we can lose vital information that is stored elsewhere. |

| **Noncompliant Code** |
| --- |
| In the code block below input is read directly and it is possible that this memory location can be written over and the information could be lost forever. |
| const std::string account\_number = "CharlieBrown42";  char user\_input[20];  std::cout << "Enter a value: ";  std::cout << "You entered: " << user\_input << std::endl;  std::cout << "Account Number = " << account\_number << std::endl; |

| **Compliant Code** |
| --- |
| In this code block the input is only read to a point and will not allow the user to write over data that is stored. |
| const std::string account\_number = "CharlieBrown42";  char user\_input[20];  std::cout << "Enter a value: ";  //Added the getline command to make sure that there are not more than 20 characters added to the user\_input variable  std::cin.getline(user\_input, 20);  std::cout << "You entered: " << user\_input << std::endl;  std::cout << "Account Number = " << account\_number << std::endl; |

| **Principles(s):** Validate Input Data: We saw in our assignment a couple of weeks ago that there were potential issues with data where we were writing over data that was already included, but it is relatively easy to make sure that the information is not written over. |
| --- |
| Heed Compiler Warnings: If we make sure that we check the compiler warnings we can see if there is a potential warning where data might be overwritten and cause issues. |
| Use Proper Quality Assurance Techniques: This means making sure that quality assurance is followed and we can check to make sure that this issue is dealt with properly. |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level (1-5)** |
| --- | --- | --- | --- | --- |
| Low | Medium | Medium (Loss of data being written over is important, but this is an issue that is not likely to be ran across in typical coding. | Medium | 4 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CppCheck | 2.9 | CppCheck’s Desktop Application | This application checks for errors specifically with the C++ language. This tool allows the developer to check and make sure that all of the errors are dealt with and gives a brief explanation as to why there is an error in the software. |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | [STD-006-CPP] | If a programmer asserts that a value is something and does not check that, the code can spit out wrong data. |

| **Noncompliant Code** |
| --- |
| The code below shows an example where a value was changed, and the user did not correct it and the data calculations at the end of the block gave us the wrong output. |
| int x = 7;  int numSquared;  //do some more code here  x= 9;  //do more code here  numSquared = x \* x;  std::cout << numSquared; |

| **Compliant Code** |
| --- |
| The code block below makes sure that the value that we entered at the beginning is the same at the end to make sure that the data that we got is correct. This makes sure that a malicious user does not change any values to cause issues with the system. |
| int x = 7;  int numSquared;  //do some more code here  x= 9;  //do more code here  assert(x == 7);  numSquared = x \* x;  std::cout << numSquared; |

| **Principles(s):** Architect Design for Security Policies: This boils down to making sure that once you have used your assert statements, that the information associated with it is dealt with properly. If the assert information is essential to the code running properly, it probably should not be used in assert statements, because they will not run as intended in future builds. |
| --- |
| Keep it Simple: By keeping assert statements simple, we can limit the complexity around these statements and limit the number of errors that might arise because of this error. |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level (1-5)** |
| --- | --- | --- | --- | --- |
| Low | High | Low-Medium (Assertions can be detrimental, but in the grand scheme of things, they are a lot less lickely then a lot of the other issues defined. | Low | 5 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CppCheck | 2.9 | CppCheck’s Desktop Application | This application checks for errors specifically with the C++ language. This tool allows the developer to check and make sure that all of the errors are dealt with and gives a brief explanation as to why there is an error in the software. |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | [STD-007-CPP] | Exceptions can be used to make sure that input or other data is valid and can keep malicious users from accessing data that they should not have access to. |

| **Noncompliant Code** |
| --- |
| This code block does not catch the error with an exception and the code will stop and crash. This can allow malicious users access to data that they really should not have. |
| func(0); //invalid function causes error |

| **Compliant Code** |
| --- |
| The code block below catches an error (in this example when x < 0) and lets the code keep running after an error is reached. |
| try{  func(0);  }  catch(exception myException){  std::cout << “Error: do something here”;  } |

| **Principles(s):** Heed Compiler Warning: This means that if the complier suggests that there is an exception that would benefit the system that it is followed. |
| --- |
| Keep it Simple: If we keep the exception simple, we can account for most of the issues that could pop up with ease and limit the stress that comes with exceptions. |
| Use Effective Quality Assurance Techniques: This is almost the definition of using exception statements; this is a quality assurance technique that will show any issues before pretty much everything else. |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level (1-5)** |
| --- | --- | --- | --- | --- |
| Medium | High | Low (These errors can be accounted for elsewhere, but by identifying them in this section, we can greatly decrease the cost of dealing with theses issues compared to other sections. | High | 2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CppCheck | 2.9 | CppCheck’s Desktop Application | This application checks for errors specifically with the C++ language. This tool allows the developer to check and make sure that all of the errors are dealt with and gives a brief explanation as to why there is an error in the software. |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Number Overflow** | [STD-008-CPP] | If a user overloads a data variable, they could cause the value to overflow and run around back to the bottom. This can cause users to make data inputs invalid and cause wrong output. |

| **Noncompliant Code** |
| --- |
| This code does not check for overflow errors and will allow a malicious user to enter data that causes an overflow. |
| T result = start;  for (unsigned long int i = 0; i < steps; ++i)  {  result += increment;  }  return result; |

| **Compliant Code** |
| --- |
| This code checks to make sure that a value does not get to big and causes issues where the user can make a number negative by adding super large numbers together. |
| // Variables for checking for an overflow  T result = start;  T resultCompare;  // Loop to do the incrimental adding  for (unsigned long int i = 0; i < steps; ++i)  {  // If loop to check if there was an overflow  resultCompare = result + increment;  // Checks if the new value calculated is lower than the original (check for overflow)  if (result > resultCompare) {  result = start;  return result;  }  // Checks to see if the new value is inf  // After reading online, the isinf() function has trouble with overloads that are not double  // Casting the result to double should fix this issue (link below)  // Source: https://stackoverflow.com/questions/61646166/how-to-resolve-fpclassify-ambiguous-call-to-overloaded-function  else if (isinf(static\_cast<double>(resultCompare))){  result = start;  return result;  }  // Computes the value if no overflow is detected  else {  result += increment;  }  } |

| **Principles(s):** Validate Input Data: Validating input is the easiest way to make sure that we do not have a numeric overflow that would lead to overflow issues. By validating that the information passed to the functions we can make sure that data entered is meant to be entered and not just an error. |
| --- |
| Apply a Secure Coding Standard: This means that everyone is on the same page about this issue. This means that there is a consistent data type that should be used when dealing with certain data (e.g. for dollar amounts it would be beneficial to use a double data value). |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level (1-5)** |
| --- | --- | --- | --- | --- |
| Medium | Low | Low – High (If this type of error came up for a banking system, it would cause shockwaves, but normally this would not be considered a big error) | Medium | 3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CppCheck | 2.9 | CppCheck’s Desktop Application | This application checks for errors specifically with the C++ language. This tool allows the developer to check and make sure that all of the errors are dealt with and gives a brief explanation as to why there is an error in the software. |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Number Underflow** | [STD-009-CPP] | When a number is subtracted to a value that is lower than possible the value might underflow back to the highest value. If someone wanted to raise their bank value to a high number, they could do that by exploiting this error. |

| **Noncompliant Code** |
| --- |
| This code does not check that the code does not underflow. In doing this a malicious user can change values to benefit themselves. |
| T result = start;  for (unsigned long int i = 0; i < steps; ++i)  {  result -= decrement;  }  return result; |

| **Compliant Code** |
| --- |
| This code block does check for underflows. It might not be an exhaustive list of checks, but it would limit the possibility for underflows. |
| // Variables for checking for an overflow  T result = start;  T resultCompare;  for (unsigned long int i = 0; i < steps; ++i){  // If loop to check if there was an overflow  resultCompare = result - decrement;  // Checks if the calculated value is greater than the prior result  if (resultCompare > result) {  result = start;  return result;  }  else if (resultCompare < 0) {  result = start;  return result;  }  // If no overflow is detected, the correct value is retuned  else {  result -= decrement;  }  }  return result; |

| **Principles(s):** Sanitize Data Sent to Other Systems: This is important for this practice, because if data is not reviewed, we can run into situations where the user creates an overflow to make bigger issues. |
| --- |
| Use Effective Quality Assurance Techniques: By using quality assurance techniques that would be advantageous for checking for errors in the system, we can limit the number of issues with overflow. This quality assurance can lead to finding overflow errors that might cause issues later in the development process. |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level (1-5)** |
| --- | --- | --- | --- | --- |
| Low | Medium | Low-High (Like the last coding standard, if this type of error came up for a banking system, it would cause shockwaves, but normally this would not be considered a big error) | Medium | 3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CppCheck | 2.9 | CppCheck’s Desktop Application | This application checks for errors specifically with the C++ language. This tool allows the developer to check and make sure that all of the errors are dealt with and gives a brief explanation as to why there is an error in the software. |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Clearing the Buffer** | [STD-010-CPP] | If there is invalid input, it is important to clear the buffer to make sure that data doesn’t overflow. I have had the system keep running in an infinite loop because I did not clear the buffer after invalid input. |

| **Noncompliant Code** |
| --- |
| This code block does not check to make sure that the buffer is cleared. If it is not cleared, input can lead to an infinite loop or a loop that runs too far and can make data appear funky. |
| //The block below checks to see if the user wants to try more investment info  std::cout << "Do you want to try different investment information? (Y/N)" << endl;  std::cin >> tryAgainChar;  //This block changes the try again variable and the program is exited  if (tryAgainChar == 'n' || tryAgainChar == 'N') {  tryAgain = false;  }  //This block does not change the try again variable and the user can input more values  else if (tryAgainChar == 'y' || tryAgainChar == 'Y') {  } |

| **Compliant Code** |
| --- |
| This code block makes sure that the buffer is cleared after invalid input so data can not be accessed without proper protocols. |
| //The block below checks to see if the user wants to try more investment info  std::cout << "Do you want to try different investment information? (Y/N)" << endl;  std::cin >> tryAgainChar;  //This block changes the try again variable and the program is exited  if (tryAgainChar == 'n' || tryAgainChar == 'N') {  tryAgain = false;  }  //This block does not change the try again variable and the user can input more values  else if (tryAgainChar == 'y' || tryAgainChar == 'Y') {  }  //This block catches if there is an error  else {  cin.clear();  cin.ignore(numeric\_limits<streamsize>::max(), '\n');  throw runtime\_error("Invalid Selection: You need to enter Y for Yes or N for No");  } |

| **Principles(s):** Architect and Design for Security Policies: By making and following policies that would make sure that the user does not enter data that might be held over by the buffer and cause data to be written over. |
| --- |
| Sanitize Data Sent to Other Systems: By looking at the data to make sure the user does not enter data that could lead to an overflow that might lead to improper data values being read. |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level (1-5)** |
| --- | --- | --- | --- | --- |
| Medium | High | Low-High (Depending on the application, this type of error could cause major issues, or it could be a minor issue like adding a “3” to the end of a string. | Medium | 3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CppCheck | 2.9 | CppCheck’s Desktop Application | This application checks for errors specifically with the C++ language. This tool allows the developer to check and make sure that all of the errors are dealt with and gives a brief explanation as to why there is an error in the software. |

### Defense-in-Depth Illustration

This illustration provides a visual representation of the defense-in-depth best practice of layered security.



### Automation

Provide a written explanation using the image provided.



Automation will be used for the enforcement of and compliance to the standards defined in this policy. Green Pace already has a well-established DevOps process and infrastructure. Define guidance on where and how to modify the existing DevOps process to automate enforcement of the standards in this policy. Use the DevSecOps diagram and provide an explanation using that diagram as context.

In the above diagram there are not too many changes that need to be implemented to make this process more automated. The first step to make this process more automated is to make the Verify and Test section (in pre-production). By automating this process, we can make it easier to plan for future issues and make plans. Additionally, this gives us a plan to check and make sure that issues identified are dealt with to make our future development process easier.

The next area that can be automated to make life easier is the monitor and detect phase. This would involve use setting up a way to check and see if there are any new issues identified that would cause issues (either in the present phase or in the future). Lastly, we can use the maintain and stabilize phase to make sure that any changes made will be maintainable for the future and that the development of an application is sustainable for the future.

### Summary of Risk Assessments

Consolidate all risk assessments into one table including both coding and systems standards, ordered by standard number.

| Rule | Severity | Likelihood | Remediation Cost | Priority | Level |
| --- | --- | --- | --- | --- | --- |
| STD-001-CPP | High | Semi-Likely | Medium | High | 2 |
| STD-002-CPP | High | Likely | High | Medium-High | 2 |
| STD-003-CPP | Medium | High | Low-High | High | 2 |
| STD-004-CPP | High | Low | High | Medium-High | 3 |
| STD-005-CPP | Low | Medium | Medium | Medium | 4 |
| STD-006-CPP | Low | High | Low-Medium | Low | 5 |
| STD-007-CPP | Medium | High | Low | High | 2 |
| STD-008-CPP | Medium | Low | Low-High | Medium | 3 |
| STD-009-CPP | Low | Medium | Low-High | Medium | 3 |
| STD-010-CPP | Medium | High | Low-Medium | Medium | 3 |

### Create Policies for Encryption and Triple A

Include all three types of encryption (in flight, at rest, and in use) and each of the three elements of the Triple-A framework using the tables provided***.***

* 1. Explain each type of encryption, how it is used, and why and when the policy applies.
  2. Explain each type of Triple-A framework strategy, how it is used, and why and when the policy applies.

Write policies for each and explain what it is, how it should be applied in practice, and why it should be used.

| 1. **Encryption** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Encryption in rest | Encryption at rest is meant to describe data that is encrypted so malicious users can not access the data that is meant to be protected. This means that important data (e.g. credit card information, addresses, etc.) that is being stored is encrypted so malicious users cannot access that information. This applies to data that is being stored locally that should be protected in case of a data breach so if that information makes it to the public eye, it will cause less damage. |
| Encryption at flight | Encryption in flight (or encryption in transit) means that data being sent to other people who need it is protected by encryption. This means that data that is being transferred will be sent with a key and encrypted so if it is intercepted in the process of transferring this information the data can not be interpreted and used by malicious individuals. This should be used for sensitive information that needs to be sent between different places; this includes between different departments or agencies along with being between different parts of a system (e.g. moving from the client to the database or vice versa). |
| Encryption in use | Encryption in use refers to data being used in a system. For example, if a password is needed, it is important to make sure that is being protected when the user enters the information in the password bar. This is important because a user can get access to this information and if they do, they could use that information in ways that it is not intended. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication refers to making sure that someone has the proper level of clearance to access information. This policy goes hand in hand with the principle of least privilege and means that the user only has access to the bare minimum that is necessary to do their job or complete the process that their account gives them. This includes user logins and the level of access that they have with their new account. |
| Authorization | Authorization is like authentication but refers to making sure that an individual has the proper clearance to make changes. Like authentication, this policy goes hand in hand with the policy of least privilege. This means that a user who needs to make changes to the database actually is authorized to make those changes (this applies to any change, but database is just an example). Lastly, this means that an individual who is not authorized will be dealt with properly. |
| Accounting | The last letter in the tiple-A framework refers to the accounting portion of the developer experience. Accounting also goes hand in hand with accountability and that means that anyone who breaks the above policies is dealt with properly and they are treated the proper consequences (whether that be positive or negative) for their actions. This also means that no single person should be responsible for one area on their own, because that could lead to potential issues or abuses of power later. |

The only item you must complete beyond this point is the Policy Version History table.

## Audit Controls and Management

Every software development effort must be able to provide evidence of compliance for each software deployed into any Green Pace managed environment.

Evidence will include the following:

* Code compliance to standards
* Well-documented access-control strategies, with sampled evidence of compliance
* Well-documented data-control standards defining the expected security posture of data at rest, in flight, and in use
* Historical evidence of sustained practice (emails, logs, audits, meeting notes)

## Enforcement

The office of the chief information security officer (OCISO) will enforce awareness and compliance of this policy, producing reports for the risk management committee (RMC) to review monthly. Every system deployed in any environment operated by Green Pace is expected to be in compliance with this policy at all times.

Staff members, consultants, or employees found in violation of this policy will be subject to disciplinary action, up to and including termination.

## Exceptions Process

Any exception to the standards in this policy must be requested in writing with the following information:

* Business or technical rationale
* Risk impact analysis
* Risk mitigation analysis
* Plan to come into compliance
* Date for when the plan to come into compliance will be completed

Approval for any exception must be granted by chief information officer (CIO) and the chief information security officer (CISO) or their appointed delegates of officer level.

Exceptions will remain on file with the office of the CISO, which will administer and govern compliance.

## Distribution

This policy is to be distributed to all Green Pace IT staff annually. All IT staff will need to certify acceptance and awareness of this policy annually.

## Policy Change Control

This policy will be automatically reviewed annually, no later than 365 days from the last revision date. Further, it will be reviewed in response to regulatory or compliance changes, and on demand as determined by the OCISO.

## Policy Version History

| Version | Date | Description | Edited By | Approved By |
| --- | --- | --- | --- | --- |
| 1.0 | 08/05/2020 | Initial Template | David Buksbaum |  |
| 1.1 | 9/18/2022 | Updated coding standards and principles | James Porter |  |
| 1.2 | 10/07/2022 | Completed the rest of the document and filled in all remaining requirements | James Porter |  |
| 1.3 | 10/9/2022 | Final edits for submission | James Porter |  |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

| Language | Acronym |
| --- | --- |
| C++ | CPP |
| C | CLG |
| Java | JAV |