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



# Green Pace Secure Development Policy

## Contents

[Overview 2](#_Toc52464053)

[Purpose 2](#_Toc52464054)

[Scope 2](#_Toc52464055)

[Module Three Milestone 2](#_Toc52464056)

[Ten Core Security Principles 2](#_Toc52464057)

[C/C++ Ten Coding Standards 3](#_Toc52464058)

[Coding Standard 1 4](#_Toc52464059)

[Coding Standard 2 5](#_Toc52464060)

[Coding Standard 3 6](#_Toc52464061)

[Coding Standard 4 7](#_Toc52464062)

[Coding Standard 5 8](#_Toc52464063)

[Coding Standard 6 9](#_Toc52464064)

[Coding Standard 7 10](#_Toc52464065)

[Coding Standard 8 11](#_Toc52464066)

[Coding Standard 9 13](#_Toc52464067)

[Coding Standard 10 14](#_Toc52464068)

[Defense-in-Depth Illustration 15](#_Toc52464069)

[Project One 15](#_Toc52464070)

[1. Revise the C/C++ Standards 15](#_Toc52464071)

[2. Risk Assessment 15](#_Toc52464072)

[3. Automated Detection 15](#_Toc52464073)

[4. Automation 15](#_Toc52464074)

[5. Summary of Risk Assessments 16](#_Toc52464075)

[6. Create Policies for Encryption and Triple A 16](#_Toc52464076)

[7. Map the Principles 17](#_Toc52464077)

[Audit Controls and Management 18](#_Toc52464078)

[Enforcement 18](#_Toc52464079)

[Exceptions Process 18](#_Toc52464080)

[Distribution 19](#_Toc52464081)

[Policy Change Control 19](#_Toc52464082)

[Policy Version History 19](#_Toc52464083)

[Appendix A Lookups 19](#_Toc52464084)

[Approved C/C++ Language Acronyms 19](#_Toc52464085)

## 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 | Validating input can consist of checking for proper input characters, blocking malicious code from being entered, and preventing special characters from being input. Validating Input Data can help prevent malicious attacks within data input fields. |
| 1. Heed Compiler Warnings | It is important to heed compiler warnings and eliminate any warnings by modifying code. |
| 1. Architect and Design for Security Policies | It is important to design for security policies. Planning for security helps prevent threat mitigation at a later stage. |
| 1. Keep It Simple | Keep it simple means reducing complexity of the whole system. Complex systems can lead to undetected and exploitable errors. |
| 1. Default Deny | Permission is defaulted to be denied and must be granted. This helps ensure unnecessary and risky permissions are not granted if not absolutely necessary. |
| 1. Adhere to the Principle of Least Privilege | Everything should be granted the lowest privilege necessary to ensure proper functioning code. This helps prevent malicious attacks utilizing unnecessarily elevated privileges. |
| 1. Sanitize Data Sent to Other Systems | It is important to sanitize any data before sending to other systems. This can help ensure attacks such as SQL injection cannot take place |
| 1. Practice Defense in Depth | Defense in depth means utilizing multiple layers of defense to prevent successful attacks. In this way, if one defensive layer fails, there is another defensive layer protecting the system. |
| 1. Use Effective Quality Assurance Techniques | Effective Quality Assurance is crucial to ensuring the security of the system. Regular checks of the system, utilizing techniques such as penetration testing, internal and external audits, and many other techniques can help ensure the security of the system. |
| 1. Adopt a Secure Coding Standard | Create a secure coding standard to follow for any language or platform being utilized. |

### 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] | Do not cast to an out-of-range enumeration value |

| **Noncompliant Code** |
| --- |
| This noncompliant code example attempts to check whether a given value is within the range of acceptable enumeration values. However, it is doing so after casting to the enumeration type, which may not be able to represent the given integer value. On a two's complement system, the valid range of values that can be represented by EnumType are [0..3], so if a value outside of that range were passed to f(), the cast to EnumType would result in an unspecified value, and using that value within the if statement results in unspecified behavior. |
| enum EnumType {    First,    Second,    Third  };    void f(**int** intVar) {    EnumType enumVar = static\_cast<EnumType>(intVar);      if (enumVar < First || enumVar > Third) {      // Handle error    }  } |

| **Compliant Code** |
| --- |
| This compliant solution checks that the value can be represented by the enumeration type before performing the conversion to guarantee the conversion does not result in an unspecified value. It does this by restricting the converted value to one for which there is a specific enumerator value. |
| enum EnumType {    First,    Second,    Third  };    void f(int intVar) {    if (intVar < First || intVar > Third) {      // Handle error    }    EnumType enumVar = static\_cast<EnumType>(intVar);  } |

**Note: Stop here for the milestone. Complete this section for Project One in Module Six.**

| **Principles(s):** Validate Input Data |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | Low | High | 2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Google C++ Testing Framework | 1.11.0 |  | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

ST

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | [STD-002-CPP] | Value-returning functions must return a value from all exit paths |

| **Noncompliant Code** |
| --- |
| This example does not return for positive input. |
| Int absolute\_value(int a) {  If (a < 0) {  Return -a;  }  } |

| **Compliant Code** |
| --- |
| All code paths now return a value. |
| Int absolute\_value(int a) {  If (a < 0) {  Return -a;  }  Return a;  } |

**Note: Stop here for the milestone. Complete this section for Project One in Module Six.**

| **Principles(s):** Validate Input Data |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Medium | Unlikely | Low | High | 3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Google C++ Testing Framework | 1.11.0 | Declare int in all possible ranges and utilize tests | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | [STD-003-CPP] | Guarantee that strings captured have sufficient storage for captured data |

| **Noncompliant Code** |
| --- |
| The input is not pre-defined in size so input could lead to buffer overflow. |
| #include <iostream>  Void f() {  Char input[16];  Std::cin >> input;  } |

| **Compliant Code** |
| --- |
| Utilize Std::cin.getline(variable, sizeOfBuffer) to capture input data. This ensures only the desired quantity of characters are captured, eliminating buffer overflow. |

|  |
| --- |
| #include <iostream>  Void f() {  Char input[16];  Std::cin.getline(input, 16);  } |

**Note: Stop here for the milestone. Complete this section for Project One in Module Six.**

| **Principles(s):** Validate Input Data |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | Low | High | 1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Google C++ Testing Framework | 1.11.0 | Develop unit test for different input lengths | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | [STD-004-CPP] | Sanitize input for SQL queries to prevent SQL injection. |

| **Noncompliant Code** |
| --- |
| This example utilizes string formatting, which exposes the code to SQL injection attacks |
| #include <iostream>  Void f() {  Char input[20];  Std::cin.getline(input, 20);  Char\* query = “SELECT ID FROM USERS WHERE NAME=” + input;  } |

| **Compliant Code** |
| --- |
| This code utilized prepared statement which eliminates the risk of SQL injection. |
| #include <iostream>  // . . .  Void f() {  Sql::PreparedStatement \*prep\_stmt    prep\_stmt = con->prepareStatement(“SELECT ID FROM USERS WERE NAME  (?)”;  prep\_stmt->setString(1, input);  prep\_stmt->execute();    delete prep\_stmt;  delete con;  } |

**Note: Stop here for the milestone. Complete this section for Project One in Module Six.**

| **Principles(s):** Sanitize Data Sent to Other Systems |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | Medium | High | 1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Google C++ Testing Framework | 1.11.0 | Develop unit tests with possible SQL injection queries and test results | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | [STD-005-CPP] | Allocate Sufficient memory for an object |

| **Noncompliant Code** |
| --- |
| An array of long is allocated and assigned to p. The code attempt to check for unsigned integer overflow. The code utilizes sizeof(int) which can fill the array and cause a heap buffer overflow. |
| #include <stdint.h>  #include <stdlib.h>    void function(size\_t len) {    long \*p;    if (len == 0 || len > SIZE\_MAX / sizeof(long)) {      /\* Handle overflow \*/    }    p = (long \*)malloc(len \* sizeof(int));    if (p == NULL) {      /\* Handle error \*/    }    free(p);  } |

| **Compliant Code** |
| --- |
| The compliant solution uses sizeof(long) to correctly size the memory allocation |
| #include <stdint.h>  #include <stdlib.h>    void function(size\_t len) {    long \*p;    if (len == 0 || len > SIZE\_MAX / sizeof(long)) {      /\* Handle overflow \*/    }    p = (long \*)malloc(len \* sizeof(long));    if (p == NULL) {      /\* Handle error \*/    }    free(p);  } |

**Note: Stop here for the milestone. Complete this section for Project One in Module Six.**

| **Principles(s):** Validate Input Data |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Medium | Low | Low | Medium | 3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | [STD-006-CPP] | Expressions used in assertions must not produce side effects |

| **Noncompliant Code** |
| --- |
| This code attempt to compare two strings, leading to side effects when assertions are removed |
| // . . .  Std::string str1 (“some text”);  Std::string str2 (“more text”);  Assert str1.compare(str2); // side effects  // . . . |

| **Compliant Code** |
| --- |
| The Boolean is decoupled from the assertion, removing the side effects in assertion. |
| // . . .  Std::string str1 (“some text”);  Std::string str2 (“more text”);  Boolean isEqual = str1.compare(str2);  Assert isEqual; // No side effect  // . . . |

**Note: Stop here for the milestone. Complete this section for Project One in Module Six.**

| **Principles(s):** Heed Compiler Warnings |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Medium | Likely | High | High | 1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Cppcheck | 2.6 | assignmentInAssert | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | [STD-007-CPP] | Guarantee exception safety |

| **Noncompliant Code** |
| --- |
| The following noncompliant code example shows a flawed copy assignment operator. The implicit invariants of the class are that the array member is a valid (possibly null) pointer and that the nElems member stores the number of elements in the array pointed to by array. The function deallocates array and assigns the element counter, nElems, before allocating a new block of memory for the copy. As a result, if the new expression throws an exception, the function will have modified the state of both member variables in a way that violates the implicit invariants of the class. Consequently, such an object is in an indeterminate state and any operation on it, including its destruction, results in undefined behavior. |
| #include <cstring>    class IntArray {    int \*array;    std::size\_t nElems;  public:    // ...      ~IntArray() {      delete[] array;    }        IntArray(const IntArray& that); // nontrivial copy constructor    IntArray& operator=(const IntArray &rhs) {      if (this != &rhs) {        delete[] array;        array = nullptr;        nElems = rhs.nElems;        if (nElems) {          array = new int[nElems];          std::memcpy(array, rhs.array, nElems \* sizeof(\*array));        }      }      return \*this;    }      // ...  }; |

| **Compliant Code** |
| --- |
| In this compliant solution, the copy assignment operator provides the strong exception safety guarantee. The function allocates new storage for the copy before changing the state of the object. Only after the allocation succeeds does the function proceed to change the state of the object. In addition, by copying the array to the newly allocated storage before deallocating the existing array, the function avoids the test for self-assignment, which improves the performance of the code in the common case. |
| #include <cstring>    class IntArray {  **int** \*array;    std::**size\_t** nElems;  public:    // ...      ~IntArray() {      delete[] array;    }      IntArray(const IntArray& that); // nontrivial copy constructor      IntArray& operator=(const IntArray &rhs) {  **int** \*tmp = nullptr;      if (rhs.nElems) {        tmp = new **int**[rhs.nElems];        std::**memcpy**(tmp, rhs.array, rhs.nElems \* sizeof(\*array));      }      delete[] array;      array = tmp;      nElems = rhs.nElems;      return \*this;    }      // ...  }; |

**Note: Stop here for the milestone. Complete this section for Project One in Module Six.**

| **Principles(s):** Adopt a Secure Coding Standard |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Unlikely | High | Low | 4 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Characters and Strings** | [STD-008-CPP] | Range check element access |

| **Noncompliant Code** |
| --- |
| In this example, the value returned may be greater than the number of elements stored in the string. |
| #include <string>    extern std::**size\_t** get\_index();    void f() {    std::string s("01234567");    s[get\_index()] = '1';  } |

| **Compliant Code** |
| --- |
| This solution checks that the value returned by get\_index() is within a valid range before calling operator[](). |
| #include <string>    extern std::**size\_t** get\_index();    void f() {    std::string s("01234567");    std::**size\_t** i = get\_index();    if (i < s.length()) {      s[i] = '1';    } else {      // Handle error    }  } |

**Note: Stop here for the milestone. Complete this section for Project One in Module Six.**

| **Principles(s):** Validate Input Data |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Medium | Unlikely | Low | Medium | 3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Google C++ Testing Framework | 1.11.0 | Develop unit test to test possible index ranges | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Random Number Generation** | [STD-009  -CPP] | Do not use std::rand() for generating pseudorandom numbers. |

| **Noncompliant Code** |
| --- |
| The following noncompliant code generates an ID with a numeric part produced by calling the rand() function. The IDs produced are predictable and have limited randomness. Further, depending on the value of RAND\_MAX, the resulting value can have modulo bias. |
| #include <cstdlib>  #include <string>    void f() {    std::string id("ID"); // Holds the ID, starting with the characters "ID" followed                          // by a random integer in the range [0-10000].    id += std::to\_string(std::**rand**() % 10000);    // ...  } |

| **Compliant Code** |
| --- |
| The C++ standard library provides mechanisms for fine-grained control over pseudorandom number generation. It breaks random number generation into two parts: one is the algorithm responsible for providing random values (the engine), and the other is responsible for distribution of the random values via a density function (the distribution). The distribution object is not strictly required, but it works to ensure that values are properly distributed within a given range instead of improperly distributed due to bias issues. This compliant solution uses the Mersenne Twister algorithm as the engine for generating random values and a uniform distribution to negate the modulo bias from the noncompliant code example. |
| #include <random>  #include <string>    void f() {    std::string id("ID"); // Holds the ID, starting with the characters "ID" followed                          // by a random integer in the range [0-10000].    std::uniform\_int\_distribution<int> distribution(0, 10000);    std::random\_device rd;    std::mt19937 engine(rd());    id += std::to\_string(distribution(engine));    // ...  } |

**Note: Stop here for the milestone. Complete this section for Project One in Module Six.**

| **Principles(s):** Adopt a Secure Coding Standard |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Medium | Unlikely | Low | High | 2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Miscellaneous** | [STD-010-CPP] | Do not return from a function declared [[noreturn]] |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, if the value 0 is passed, control will flow off the end of the function, resulting in an implicit return and undefined behavior. |
| #include <cstdlib>    [[noreturn]] void f(int i) {  if (i > 0)  throw "Received positive input";  else if (i < 0)  std::exit(0);  } |

| **Compliant Code** |
| --- |
| In this compliant solution, the function does not return on any code path. |
| #include <cstdlib>    [[noreturn]] void f(int i) {    if (i > 0)      throw "Received positive input";    std::exit(0);  } |

**Note: Stop here for the milestone. Complete this section for Project One in Module Six.**

| **Principles(s):** Adopt a Secure Coding Standard |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Unlikely | Low | High | 2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Cppcheck | 2.6 | Check Function usage | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

### Defense-in-Depth Illustration

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



## Project One

There are seven steps outlined below that align with the elements you will be graded on in the accompanying rubric. When you complete these steps, you will have finished the security policy.

### Revise the C/C++ Standards

You completed one of these tables for each of your standards in the Module Three milestone. In Project One, add revisions to improve the explanation and examples as needed. Add rows to accommodate additional examples of compliant and noncompliant code. Coding standards begin on the security policy.

### Risk Assessment

Complete this section on the coding standards tables. Enter high, medium, or low for each of the headers, then rate it overall using a scale from 1 to 5, 5 being the greatest threat. You will address each of the seven policy standards. Fill in the columns of severity, likelihood, remediation cost, priority, and level using the values provided in the appendix.

### Automated Detection

Complete this section of each table on the coding standards to show the tools that may be used to detect issues. Provide the tool name, version, checker, and description. List one or more tools that can automatically detect this issue and its version number, name of the rule or check (preferably with link), and any relevant comments or description—if any. This table ties to a specific C++ coding standard.

### 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.

Automation will need to be added the verify and test phase. This will allow the different automated detection and enforcement of compliance to be used before the project enters the production phase, where any vulnerabilities can be exploited. Unit tests will need to be written to test for standards as well as Cppcheck analysis performed on the code. Any issues will need to be addressed before moving to production.

### 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 | Likely | Low | High | 2 |
| STD-002-CPP | Medium | Unlikely | Low | High | 3 |
| STD-003-CPP | High | Likely | Low | High | 1 |
| STD-004-CPP | High | Likely | Medium | High | 1 |
| STD-005-CPP | Medium | Unlikely | Low | Medium | 3 |
| STD-006-CPP | Medium | Unlikely | Low | Medium | 3 |
| STD-007-CPP | Low | Unlikely | High | Low | 4 |
| STD-008-CPP | Medium | Unlikely | Low | Medium | 3 |
| STD-009-CPP | Medium | Unlikely | Low | High | 2 |
| STD-010-CPP | High | Unlikely | Low | High | 2 |

### 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 | Data in storage is encrypted, ensuring any data breach will be unreadable |
| Encryption at flight | Any data being sent to other systems or external services will be encrypted during transmission utilizing secure encryption keys. This will ensure any data that is captured mid-flight will not be accessible |
| Encryption in use | Data being used will only be unencrypted to verify accuracy, but will not be stored, returned, or transmitted to any other system without being first encrypted. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication is ensuring the user or system is who or what it says it is. This can include verifying users with a username and password. |
| Authorization | Authorization is ensuring the user is authorized to do what they request to do. The lowest required authorization should be allowed to ensure access that is not necessary is not given. It is important to maintain a proper user level of access and not give unnecessary permissions to when adding new users. |
| Accounting | Accounting is a measure of what a user does during access such as changes to the database and what files are accessed by users. This allows user activity to be audited to check for what was done and requested and what was granted. |

**\***Use this checklist for the Triple A to be sure you include these elements in your policy:

* User logins
* Changes to the database
* Addition of new users
* User level of access
* Files accessed by users

### Map the Principles

Map the principles to each of the standards, and provide a justification for the connection between the two. In the Module Three milestone, you added definitions for each of the 10 principles provided. Now it’s time to connect the standards to principles to show how they are supported by principles. You may have more than one principle for each standard, and the principles may be used more than once. Principles are numbered 1 through 10. You will list the number or numbers that apply to each standard, then explain how each of these principles supports the standard. This exercise demonstrates that you have based your security policy on widely accepted principles. Linking principles to standards is a best practice.

**NOTE:** Green Pace has already successfully implemented the following:

* Operating system logs
* Firewall logs
* Anti-malware logs

**STD-001-CPP:** Principle 1 – Validate input data. It is important to validate input data to ensure that casting of enumeration value is not out-of-range

**STD-002-CPP:** Principle 10 – Adapt a secure Coding Standard. Ensuring all exit paths return a value is part of secure coding to ensure all functions return a value and do not cause unexpected behavior.

**STD-003-CPP:** Principle 1 – Validate input data. It is important to check captured strings have sufficient storage for captured data. This ensures there will not be a buffer overflow.

**STD-004-CPP:** Principle 7 – Sanitize Data Sent to Other Systems. Ensuring data sent to SQL server is properly sanitized is crucial to protect against SQL injection attacks

**STD**-**005-CPP:** Principle 1 – All data being input into a function should be validated to ensure proper memory storage is used. This prevents against overflow errors

**STD-006-CPP:** Principle 2 – Heed compiler warnings. Improper uses of assertions will be flagged by compiler warnings. It is important to heed these to ensure code is functioning as intended.

**STD-007-CPP:** Principle 10 – Adopt a secure Coding Standard.

**STD-008-CPP:** Principle 1 – Validate Input Data. All data being input into the function should be validated before use. In this case, the value of the element should be checked before being accessed.

**STD-009-CPP:** Principle 10 – Adopt a secure Coding Standard. Utilizing rand() can have a modulo bias, this can lead to predictable results.

**STD-010-CPP:** Principle 10 – Adopt a secure coding standard. It is important that functions declared noreturn do not ever result in an implicit return. This could lead to undesired and exploitable behavior.

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 | 10/10/2021 | Completed Document | Nicholas Newlin | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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