**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 any input data before handling is essential for security. This includes verifying that the input follows the anticipated formats, types, and ranges to avoid malicious data and prevent it from causing unexpected actions or vulnerabilities |
| 1. Heed Compiler Warnings | Compiler warnings are possible issues within the code that could be a warning of potential security vulnerabilities. Overlooking these warnings can lead to overlooking security risks. Developers should refer to the warnings and solve the issues to protect the codes’ reliability and security. |
| 1. Architect and Design for Security Policies | Security should be incorporated into the system’s architecture and design in the start of an applications production. This involves recognizing and employing security policies to preserve the system and make sure it is robust and strong to threats and continues to display the best practices for protecting data and resources. |
| 1. Keep It Simple | Simplicity of design and implementation benefits an application by decreasing the probability of security vulnerabilities. A simple system is easier to understand, review, and maintain, which lowers the chances of errors showing up and makes it simpler to protected. |
| 1. Default Deny | By default, deny access to resources and only allow permissions when needed. This principal safeguards against unauthorized access are prevented by default, reducing the risk of sensitive data or functions being exposed to unauthorized users. |
| 1. Adhere to the Principle of Least Privilege | Users and actions should be authorized only the minimal level of access needed to complete their tasks and responsibilities. This helps prevent potential damage from unintended or malicious actions and lowers the chance of security breaches. |
| 1. Sanitize Data Sent to Other Systems | When moving data to outside systems, it's necessary to sanitize or “cleanse” the data to prevent security risks such as data leaks or injection attacks. This involves confirming that data is cleaned, properly formatted, and free from destructive contents. |
| 1. Practice Defense in Depth | Employ and use multiple layers of security features to protect a system. This approach helps guarantee that if one layer fails, other layers will continue to provide protection to the system and data, improving overall security and resistance against attacks. |
| 1. Use Effective Quality Assurance Techniques | Employing thorough quality assurance practices, such as vulnerability assessments, automated testing, and code reviews, benefits identifying and addressing potential security issues before they develop into problems within the production environments. |
| 1. Adopt a Secure Coding Standard | Establishing and adhering secure coding standards to help guide developers in writing code that’s robust against common security threats. These standards present best practices and principles to safeguard code and makes the code secure and less prone to vulnerabilities. |

### 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] | Use the most applicable and specialized data type for the specific circumstance. Assure consistent use of data types within the codebase. Avoid unnecessary type conversions and casts. Choose data types that improve memory handling. Prevent using obscure or exceedingly complicated data type declarations. Employ data types that enhance code readability and maintainability. Prevent unnecessary type conversions and casts. Avoid using generic data types when a certain type is available and works better. |

| **Noncompliant Code** |
| --- |
| int should be used as an alternative of double for whole numbers to enhance precision and memory usage. Keep away from useless type conversions to maintain simplicity and stop prospective performance issues. |
| // Using double for a whole number  double itemCount = 20.0;  // Converting between types unnecessarily  int apples = 12;  double doubleApples = (double)apples; |

| **Compliant Code** |
| --- |
| Using int for whole numbers improves accuracy and memory productivity, as int uses less memory than double and prevents potential floating-point precision concerns. Avoiding redundant type conversions protects simplicity and improves performance by reducing the transparency of type casting. |
| // Use whole numbers  int itemCount = 20;  // Directly use the appropriate type  int apples = 12; |

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

| **Principles(s):** [Name the principle and explain how it maps to this standard.] |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

**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 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | [STD-002-CPP] | Applying strict validation rules and consistency checks to guarantee all data values used within the codebase are validated and regularly checked to avoid mistakes, maintain data reliability, and improve the overall consistency of the application. |

| **Noncompliant Code** |
| --- |
| Code does not validate the input before processing which can lead to unexpected behaviors and security vulnerabilities. |
| int IDLength = userInput(); // Assumes input is valid  if (userID < 0) {  // Handle invalid ID, but no validation when assigning  } |

| **Compliant Code** |
| --- |
| Input data is validated before being used, ensuring that only valid values are processed. |
| int IDLength = userInput();  if (IDLength < 5 || IDLength > 7 ) {  throw new IllegalArgumentException("ID is invalid length");  }  else {  // use IDLength because it has been validated  } |

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

| **Principles(s):** [Name the principle and explain how it maps to this standard.] |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

**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 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | [STD-003-CPP] | Guarantee string handling in the codebase is done appropriately, focusing on validation, accurate usage, and preventing vulnerabilities linked with string operations. String correctness is needed for avoiding problems such as buffer overflows, invalid data processing, and security vulnerabilities. |

| **Noncompliant Code** |
| --- |
| Code handles strings in a risky manner, which could lead to buffer overflows and security vulnerabilities. |
| char buffer[13];  strcpy(buffer, "This string is too long"); |

| **Compliant Code** |
| --- |
| Code uses protected string handling functions and includes validation to guarantee that the string operations do not go over buffer limits. |
| void safeCopy(char\* dest, const char\* src, size\_t destSize) {  if (strlen(src) >= destSize) {  throw std::runtime\_error("String is too long");  }  strncpy(dest, src, destSize - 1);  dest[destSize - 1] = '\0'; // Ensure null-termination  }  char buffer[13];  saferCopy(buffer, "Short string", sizeof(buffer)); |

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

| **Principles(s):** [Name the principle and explain how it maps to this standard.] |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

**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 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | [STD-004-CPP] | Prevent SQL Injection |

| **Noncompliant Code** |
| --- |
| Code is vulnerable to SQL injection because it creates SQL queries by directly using input without proper sanitization or parameterization. |
| std::string userInput = getUserInput(); // gets user input  std::string query = "SELECT \* FROM users WHERE username = '" + userInput + "';";  executeQuery(query); |

| **Compliant Code** |
| --- |
| Code uses parameterized queries to manage user input. This method helps prevents SQL injection by ensuring that user input is used as data rather than executable SQL code. |
| // gets user input  std::string userInput = getUserInput();  std::string query = "SELECT \* FROM users WHERE username = ?;";  PreparedStatement stmt = connection.prepareStatment(query);  stmt.setString(1, userInput);  executeQuery(stmt); |

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

| **Principles(s):** [Name the principle and explain how it maps to this standard.] |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

**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 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | [STD-005-CPP] | Handles and detects memory allocation errors. |

| **Noncompliant Code** |
| --- |
| Failure to check return value of a memory allocation function |
| char\* buffer = (char\*)malloc(1032);  strcpy(buffer, "Important data"); |

| **Compliant Code** |
| --- |
| Checks return value of memory allocation functions to guarantee the allocation was successful |
| char\* buffer = (char\*)malloc(1032);  if (buffer == nullptr) {  // Handle allocation failure  printf(stderr, "Memory allocation has failed\n");  exit(EXIT\_FAILURE);  }  strcpy(buffer, "Important data"); |

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

| **Principles(s):** [Name the principle and explain how it maps to this standard.] |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

**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] | A tool used to enforce assumptions and validate internal program logic throughout development and testing. |

| **Noncompliant Code** |
| --- |
| Using assertions in a way that affects the program's behavior or modifies state or relying on assertions for runtime error handling as an alternative of using proper exception handling or error checking. |
| void process(int value) {  assert(value >= 0); // Incorrect - value can be negative in normal operation  // Processing code  } |

| **Compliant Code** |
| --- |
| Use assertions to validate assumptions that should continually hold true during development and testing. |
| void process(int value) {  assert(value >= 0); // Valid - should always be non-negative based on program logic  } |

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

| **Principles(s):** [Name the principle and explain how it maps to this standard.] |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

**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 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | [STD-007-CPP] | Guarantees that exceptions are used appropriately to manage error conditions without interrupting the program's flow or indicating to undefined behavior. |

| **Noncompliant Code** |
| --- |
| Improper use of exceptions can lead to performance issues or interrupt program flow if exceptions are used for control flow or if they are not caught properly, leading to possible crashes or undefined behavior. |
| void process(int value) {  try {  if (value < 0) {  throw std::runtime\_error("Negative value");  }  // Processing code  } catch (...) {  // Catch-all block that may handle unintended exceptions  }  } |

| **Compliant Code** |
| --- |
| Use exceptions for managing error conditions that are not assumed to occur during normal execution. |
| void process(int value) {  if (value < 0) {  throw std::invalid\_argument("Negative value is not allowed");  }  // Processing code  }  void safeProcess(int value) {  try {  process(value);  } catch (const std::invalid\_argument& e) {  // Handle specific exception  std::cerr << "Error: " << w.what() << std::endl;  }  } |

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

| **Principles(s):** [Name the principle and explain how it maps to this standard.] |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

**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** |
| --- | --- | --- |
| Code Modularity | [STD-008-CPP] | Promotes reusability, ease of testing, and maintenance, allowing for better organization and separation of concerns in the codebase. |

| **Noncompliant Code** |
| --- |
| A lack of modularity results in monolithic code that is difficult to maintain, test, or reuse. |
| void processOrder(int orderId, std::string customerName) {  // Retrieve order details from database  // Process payment  // Send confirmation email  // Update inventory  } |

| **Compliant Code** |
| --- |
| Modular code separates responsibilities into distinct functions or classes, promoting reusability, ease of testing, and maintainability. |
| void retrieveOrderDetails(int ordersId);  void processPayment(int ordersId);  void sendConfirmEmail(std::string customersName);  void updateInventy(int ordersId);  void processOrder(int orderId, std::string customersName) {  retrieveOrderDetail(orderId);  processPayment(orderId);  sendConfirmEmail(customersName);  updateInventory(orderId);  } |

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

| **Principles(s):** [Name the principle and explain how it maps to this standard.] |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

**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 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Resource Management | [STD-009-CPP] | Guarantee proper management of resources such as memory, file handles, and network connections. |

| **Noncompliant Code** |
| --- |
| Inappropriate management of resources such as memory, file handles, or network connections can lead to resource leaks, crashes, and undefined behavior. |
| void processData() {  int\* data = new int[100]; // Memory allocated but not deallocated  // Process data  // Code execution ends without freeing the memory  } |

| **Compliant Code** |
| --- |
| Appropriate resource management involves ensuring that all resources, such as memory, file handles, and network connections, are properly allocated and released. |
| void processUserData() {  std::unique\_ptr<int[]> data(new int[100]); // memory is freed when out of scope  // Process data  // Memory is deallocated when the function exits  } |

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

| **Principles(s):** [Name the principle and explain how it maps to this standard.] |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

**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** |
| --- | --- | --- |
| Naming Conventions | [STD-010-CPP] | Regular naming conventions for variables, functions, classes, and other identifiers. |

| **Noncompliant Code** |
| --- |
| Inconsistent or unclear naming conventions can make code difficult to read and maintain, leading to confusion and potential errors. |
| int x; // Ambiguous variable name  void processData(); // Function name lacks specificity  class myClass {}; |

| **Compliant Code** |
| --- |
| Use clear and consistent naming conventions for variables, functions, classes, and other identifiers. |
| int userName; // Descriptive variable name  void processUserData(); // Function name specifies the purpose  class UserProfile {}; |

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

| **Principles(s):** [Name the principle and explain how it maps to this standard.] |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

**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.] |

### 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.

[Insert your written explanations here.]

### 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 | Unlikely | Medium | High | 2 |
| [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.] | [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.] | [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.] |
| [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.] | [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.] |

### 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 at rest | [Insert text.] |
| Encryption in flight | [Insert text.] |
| Encryption in use | [Insert text.] |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | [Insert text.] |
| Authorization | [Insert text.] |
| Accounting | [Insert text.] |

**\***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

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 |  |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [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 |