**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 8](#_Toc52464062)

[Coding Standard 5 10](#_Toc52464063)

[Coding Standard 6 11](#_Toc52464064)

[Coding Standard 7 13](#_Toc52464065)

[Coding Standard 8 15](#_Toc52464066)

[Coding Standard 9 17](#_Toc52464067)

[Coding Standard 10 19](#_Toc52464068)

[Defense-in-Depth Illustration 21](#_Toc52464069)

[Project One 21](#_Toc52464070)

[1. Revise the C/C++ Standards 21](#_Toc52464071)

[2. Risk Assessment 21](#_Toc52464072)

[3. Automated Detection 21](#_Toc52464073)

[4. Automation 21](#_Toc52464074)

[5. Summary of Risk Assessments 22](#_Toc52464075)

[6. Create Policies for Encryption and Triple A 22](#_Toc52464076)

[7. Map the Principles 23](#_Toc52464077)

[Audit Controls and Management 24](#_Toc52464078)

[Enforcement 24](#_Toc52464079)

[Exceptions Process 24](#_Toc52464080)

[Distribution 25](#_Toc52464081)

[Policy Change Control 25](#_Toc52464082)

[Policy Version History 25](#_Toc52464083)

[Appendix A Lookups 25](#_Toc52464084)

[Approved C/C++ Language Acronyms 25](#_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 | Definition: Input validation ensures that all data received from untrusted sources (e.g., user input) is properly checked before being processed. This prevents malicious inputs such as SQL injections, cross-site scripting (XSS), or buffer overflows.  Example: Using a regular expression to validate email addresses or ensuring that numerical fields only contain numbers. |
| 1. Heed Compiler Warnings | Definition: Compiler warnings indicate potential issues or risky practices in the code, such as unused variables or risky function calls. Ignoring them could introduce vulnerabilities.  Example: Fixing unused variable warnings or addressing deprecated function usage in your code. |
| 1. Architect and Design for Security Policies | Definition: Secure software should be designed from the ground up to minimize security risks. This includes understanding potential vulnerabilities in the system and designing features that mitigate those risks.  Example: Using secure design patterns like layered architecture or avoiding hard-coded credentials. |
| 1. Keep It Simple | Definition: Complex code is more difficult to secure and maintain. Striving for simplicity in design reduces the chance of introducing security vulnerabilities.  Example: Avoiding overly complicated logic that can lead to hidden vulnerabilities or mistakes. |
| 1. Default Deny | Definition: The principle of default deny ensures that access is not granted unless explicitly allowed. This minimizes the attack surface and prevents unauthorized access.  Example: Denying access to all ports or services by default, and opening only those that are explicitly required. |
| 1. Adhere to the Principle of Least Privilege | Definition: Every user and system should have the minimum level of access necessary to perform its function. Limiting privileges helps reduce the potential damage from a breach.  Example: Granting users access only to the specific files and systems they need for their job. |
| 1. Sanitize Data Sent to Other Systems | Definition: Data sent between systems should be sanitized to prevent it from carrying malicious payloads, such as SQL injection or XSS scripts.  Example: Escaping special characters in data before passing it to a database or another service. |
| 1. Practice Defense in Depth | Definition: Layered security strategies ensure that if one defense fails, others will still protect the system. This reduces the risk of a single point of failure.  Example: Using firewalls, encryption, and access controls together to protect sensitive data. |
| 1. Use Effective Quality Assurance Techniques | Definition: Continuous testing and reviewing of code (through methods like code reviews, unit testing, and automated testing) can help identify and address vulnerabilities before they are exploited.  Example: Writing unit tests for every function and performing static analysis of code before deployment. |
| 1. Adopt a Secure Coding Standard | Definition: Adhering to established coding standards ensures that code is written securely from the start. This includes practices such as avoiding buffer overflows and ensuring that security flaws are avoided.  Example: Following a standard like the SEI CERT C++ Coding Standard would serve as a great example of an established standard to adhere to. |

### 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] | Choosing the appropriate data type ensures efficient memory usage and reduces the risk of bugs or overflow vulnerabilities. Using incorrect or overly general data types (like using an int when a char is more appropriate) can lead to performance issues or overflow errors. |

| **Noncompliant Code** |
| --- |
| Using an incorrect data type that could result in overflow or loss of data. |
| int num = 30000;  char smallNum = num; // Noncompliant: 'num' overflows 'smallNum' |

| **Compliant Code** |
| --- |
| Using the correct data type that aligns with the expected data range. |
| int num = 30000;  short smallNum = num; // Compliant: 'smallNum' can hold the value of 'num' |

**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] | It's essential to ensure that data values fall within the expected range to prevent vulnerabilities like buffer overflows or logic errors that could result in security breaches or unexpected behavior. |

| **Noncompliant Code** |
| --- |
| Failing to validate input values properly. |
| int age = userInput; // Noncompliant: No validation of user input for age |

| **Compliant Code** |
| --- |
| Properly validating input to ensure it meets the expected range or format. |
| int age = userInput;  if (age < 0 || age > 120) {  // Handle invalid age  } |

**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] | Buffer overflows and memory corruption often occur when strings are handled improperly. Secure handling of strings prevents these vulnerabilities by ensuring that string operations do not exceed allocated memory. |

| **Noncompliant Code** |
| --- |
| Failing to properly handle string sizes, leading to buffer overflows. |
| char buffer[10];  strcpy(buffer, "A very long string!"); // Noncompliant: buffer overflow risk |

| **Compliant Code** |
| --- |
| Using safer functions like strncpy() to prevent buffer overflows. |
| char buffer[10];  strncpy(buffer, "A very long string!", sizeof(buffer) - 1); // Compliant: prevents overflow  buffer[sizeof(buffer) - 1] = '\0'; // Ensure null termination |

**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] | SQL injection vulnerabilities occur when untrusted data is used directly in SQL queries. Attackers can manipulate queries to execute arbitrary SQL, compromising the integrity and security of the database. Proper use of parameterized queries prevents this risk. |

| **Noncompliant Code** |
| --- |
| Concatenating untrusted input directly into SQL queries, which opens the door for SQL injection. |
| std::string query = "SELECT \* FROM users WHERE username = '" + username + "' AND password = '" + password + "';";  // Noncompliant: Directly using user input in SQL query |

| **Compliant Code** |
| --- |
| Using parameterized queries (prepared statements) to avoid SQL injection. |
| std::string query = "SELECT \* FROM users WHERE username = ? AND password = ?";  // Compliant: Using prepared statement with placeholders  // Assuming you have a database connection that supports prepared statements |

**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] | Improper memory management can lead to vulnerabilities such as buffer overflows, use-after-free, or memory leaks. Using secure memory management techniques, such as using bounds-checking functions, mitigates these risks. |

| **Noncompliant Code** |
| --- |
| Failing to properly allocate or deallocate memory, which can lead to memory leaks or accessing invalid memory. |
| char\* buffer = new char[100];  // Noncompliant: Memory is allocated but never freed, leading to a memory leak |

| **Compliant Code** |
| --- |
| Properly managing memory by ensuring it is freed after use. |
| char\* buffer = new char[100];  // Compliant: Memory is deallocated after use  delete[] 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 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | [STD-006-CPP] | Assertions are used to check for conditions that should never occur in a program. They help catch errors early in the development process, preventing invalid states from propagating into production environments. |

| **Noncompliant Code** |
| --- |
| Skipping validity checks for conditions that could lead to undefined behavior or crashes. |
| int\* ptr = nullptr;  \*ptr = 42; // Noncompliant: Dereferencing a null pointer without checking for validity |

| **Compliant Code** |
| --- |
| Using assertions to ensure conditions hold true during runtime. |
| int\* ptr = nullptr;  assert(ptr != nullptr); // Compliant: Ensures ptr is not null before dereferencing  \*ptr = 42; |

**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] | Proper exception handling ensures that errors are caught and handled appropriately without crashing the application. Unchecked exceptions can lead to system instability or security vulnerabilities. |

| **Noncompliant Code** |
| --- |
| Not using exception handling or improper handling of exceptions. |
| // Noncompliant: Not catching any exceptions, which might cause an app crash  int divide(int a, int b) {  return a / b;  } |

| **Compliant Code** |
| --- |
| Properly catching and handling exceptions to ensure graceful error handling. |
| try {  int result = divide(a, b);  } catch (const std::exception& e) {  std::cerr << "Exception caught: " << e.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** |
| --- | --- | --- |
| Input Length Validation | [STD-008-CPP] | Insufficient validation of input length can lead to buffer overflows, data truncation, and performance issues. Ensuring that input lengths are validated properly prevents these types of vulnerabilities. |

| **Noncompliant Code** |
| --- |
| Input length is not validated, allowing for potential overflows or improper memory usage. |
| char userInput[10];  strcpy(userInput, longString); // Noncompliant: String may exceed buffer size |

| **Compliant Code** |
| --- |
| Checking the input length before processing it ensures that it will fit within the allocated buffer. |
| char userInput[10];  if (strlen(longString) < sizeof(userInput)) {  strcpy(userInput, longString); // Compliant: Safe copy with length check  } |

**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** |
| --- | --- | --- |
| Secure File Handling | [STD-009-CPP] | Improper handling of files (such as inadequate permission checks or not sanitizing file paths) can lead to unauthorized access, file corruption, or other vulnerabilities. Secure file handling ensures that files are accessed in a controlled manner. |

| **Noncompliant Code** |
| --- |
| Opening files without proper permission checks or validation. |
| std::ifstream file("userData.txt"); // Noncompliant: No permission checks for accessing sensitive files |

| **Compliant Code** |
| --- |
| Ensuring that file access is validated and permissions are checked before performing any file operations. |
| std::ifstream file("userData.txt");  if (file && hasPermission(file)) { // Compliant: Check for permissions before opening file  // Proceed with file operations  } |

**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** |
| --- | --- | --- |
| Validate Time and Date Inputs | [STD-010-CPP] | Time and date-related vulnerabilities, such as improper validation of user input or system clocks, can result in incorrect calculations, security flaws, and incorrect access controls. Ensuring that time and date inputs are validated is crucial for maintaining system integrity. |

| **Noncompliant Code** |
| --- |
| Accepting date or time values without validation can cause logical errors or security issues, especially with time-based access control. |
| struct tm timeStruct;  timeStruct.tm\_year = 2025; // Noncompliant: No check to ensure the year is valid |

| **Compliant Code** |
| --- |
| Properly validating and handling time and date inputs ensures they are within valid ranges. |
| struct tm timeStruct;  if (isValidYear(2025)) { // Compliant: Validate year input  timeStruct.tm\_year = 2025;  } |

**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 |