**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 | To validate any data that comes from outside the program. This may include, but is not limited to command line arguments, network interfaces, environmental variables, and user-controlled files. |
| 1. Heed Compiler Warnings | To compile your code, using the highest warning level available for your compiler, and to resolve any warnings that may occur by modifying the code set. |
| 1. Architect and Design for Security Policies | To create a software architecture and design for your software to implement and enforce the most up to date security policies |
| 1. Keep It Simple | To keep the design as simple and small as possible to decrease the likelihood that errors will be made in their implementation, configuration, and use. |
| 1. Default Deny | To base access decisions on permission rather than exclusion |
| 1. Adhere to the Principle of Least Privilege | To adhere to the process of execution with the least number of privileges necessary to complete the job. |
| 1. Sanitize Data Sent to Other Systems | To sanitize all data passed to complex subsystems such as command shells, relational databases, and commercial off-the-shelf components. |
| 1. Practice Defense in Depth | To mitigate risk by utilizing multiple defensive strategies, so that if one layer of defense turns out to be inefficient for the task, another layer of defense can prevent a security flaw from becoming an exploitable. |
| 1. Use Effective Quality Assurance Techniques | To ensure good quality assurance techniques, of which can be effective in identifying and eliminating vulnerabilities. These would include fuzz testing, penetration testing, and source code audits. |
| 1. Adopt a Secure Coding Standard | To develop and/or apply a secure coding standard for your target development language, platform, and audience. |

## 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] | Naming Conventions Signed & Unsigned Integers |

| **Noncompliant Code** |
| --- |
| The noncompliant code has incorrect uses of secure coding practices, which end up as syntax errors in the 3rd line, in which the program would incorrectly compute the mathematical operation. Although this will compile and run, it will not give correct results due to the faulty calculation method. |
| int target = -5; int num = 3;  target =- num; // Noncompliant; target = -3.  target =+ num; // Noncompliant; target = 3 |

| **Compliant Code** |
| --- |
| This compliant code has fixed the syntax error that gave the incorrect answer, and instead, gave the expected answer. |
| int target = -5; int num = 3;  target = -num; // Compliant; target += num; |

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

|  |
| --- |
| **Principles(s):** Rule 5 |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| SonarQube | 9.5 | https://rules.sonarsource.com/cpp/type/Bug?search=The%20use%20of%20operators%20pairs%20%28%3D%2B%29%20where%20the%20reversed%2C%20single%20operator%20was%20meant%20%28%2B%3D%29%20will%20compile%20and%20run%2C%20but%20not%20produce%20the%20expected%20results. | SonarQube conducts static testing to review all code provided to ensure nothing is breaking any rules inset into the program. |

### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | [STD-002-CPP] | Data values are the value stored in variables and objects throughout the code. |

| **Noncompliant Code** |
| --- |
| In C++, the variables in memory need to be released or else they can be retrieved at later times by any person/hacker. Below is an example. |
| int fun() {  char\* name = (char \*) malloc (size);  if (!name) {  return 1;  }  return 0; // Noncompliant, memory pointed by "name" has not been released } |

| **Compliant Code** |
| --- |
| The code below has fixed the error and has now freed the variable from memory in the correct way. |
| int fun() {  char\* name = (char \*) malloc (size);  if (!name) {  return 1;  }  free(name);  return 0; } |

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

|  |
| --- |
| **Principles(s):** Rule 1 |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Blocker | Semi-Likely | Medium-High | High | 5 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| SonarQube | 9.5 | https://rules.sonarsource.com/cpp/RSPEC-1232?search=Free%20memory | SonarQube conducts static testing to review all code provided to ensure nothing is breaking any rules inset into the program. |

### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | [STD-003-CPP] | Strings are some of the most commonly used variables. |

| **Noncompliant Code** |
| --- |
| The string below has concatenated of string literals with different encodings. This method will not be supported in C++ soon. |
| wchar\_t n\_array[] = "Hello" L"World"; // Noncompliant wchar\_t w\_array[] = L"Hello" "World"; // Noncompliant |

| **Compliant Code** |
| --- |
| Below is a correct version of combining strings that is supported by the current C++ standards. |
| char\_t n\_array[] = "Hello" "World"; // Compliant wchar\_t w\_array[] = L"Hello" L"World"; // Compliant |

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

|  |
| --- |
| **Principles(s):** Rule 5 |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Major | Unlikely | Small | Major | 1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| SonarQube | 9.5 | https://rules.sonarsource.com/cpp/RSPEC-817 | SonarQube conducts static testing to review all code provided to ensure nothing is breaking any rules inset into the program. |

### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | [STD-004-CPP] | Injection happens when a user implements code into the project that wasn’t initially designed to be implemented at that location. |

| **Noncompliant Code** |
| --- |
| Below is an example of an overflow that could be exploited to inject code into an SQL database. |
| char buffer[10]; scanf("%s", buffer); // Noncompliant - will overflow when a word longer than 9 characters is entered |

| **Compliant Code** |
| --- |
| Below is the compliant code that will not cause an overflow. |
| char buffer[10]; scanf("%9s", buffer); // Compliant - will not overflow |

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

|  |
| --- |
| **Principles(s):** Rule 5 |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Major | Likely | High | Major | 5 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| SonarQube | 9.5 | SonarQube is already set to check for overflow and underflow problems | SonarQube conducts static testing to review all code provided to ensure nothing is breaking any rules inset into the program. |

### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | [STD-005-CPP] | Memory leaks are some of the most common data leak exploits. |

| **Noncompliant Code** |
| --- |
| In C++ 17 and later, using “char” to negate bytes will cause errors since “char” allows for mathematical operations. |
| void handleFirstByte(char\* byte); void f(int\* i) {  char\* c = reinterpret\_cast<char\*>(i); // Noncompliant  handleFirstByte(c); } unsigned char negate(unsigned char byte) {  return ~byte; // Noncompliant } |

| **Compliant Code** |
| --- |
| This compliant code format is more secure by not allowing gaps for injecting expressions. |
| void handleFirstByte(std::byte\* byte); void f(int\* i) {  std::byte\* byte = reinterpret\_cast<std::byte\*>(i); // Compliant  handleFirstByte(byte); } std::byte negate(std::byte byte) {  return ~byte; // Compliant } |

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

|  |
| --- |
| **Principles(s):** Rule 6 |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Major | Likely | High | Major | 5 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| SonarQube | 9.5 | https://rules.sonarsource.com/cpp/RSPEC-6022 | SonarQube conducts static testing to review all code provided to ensure nothing is breaking any rules inset into the program. |

### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | [STD-006-CPP] | Method to for standard equals test |

| **Noncompliant Code** |
| --- |
| The code below is an example of how not to utilize “assert” for your programming needs. |
| assert(this.isTrue == this.isTrue) //noncompliant |

| **Compliant Code** |
| --- |
| This compliant code is a proper method to utilize the “assert” method to verify what was intended to be the outcome. |
| assert(myObject.isTrue == True) //compliant |

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

|  |
| --- |
| **Principles(s):** Rule 8 |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Low | Medium | Low | Low | 1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| SonarQube | 9.5 | Using SonarCloud or SonarQube you can set your own rules to ensure this method of testing doesn’t happen. | SonarQube conducts static testing to review all code provided to ensure nothing is breaking any rules inset into the program. |

### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | [STD-007-CPP] | Exception handling is when you expect and catch errors within a program. |

| **Noncompliant Code** |
| --- |
| This noncompliant code is an example of using a pointer as an exceptions in which errors are likely to return. |
| class E { /\* Implementation \*/}; E globalException; void fn ( int i ){  if ( i > 10 ) {  throw ( &globalException); // Noncompliant, the catch is supposed not to delete the pointer  }  else {  throw (new E ); // Noncompliant, the catch is supposed to delete the pointer  } } |

| **Compliant Code** |
| --- |
| This compliant code does not utilize a pointer to point to a new exception. Instead, utilizes the original exception. |
| class E { /\* Implementation \*/}; E globalException; void fn ( int i ){  if ( i > 10 ) {  throw ( globalException); // Throws a copy of the global variable } else {  throw (E{} ); // Throws a new object  } } |

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

|  |
| --- |
| **Principles(s):** Rule 8 |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Critical | Unlikely | Medium | High | 3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| SonarQube | 9.5 | https://rules.sonarsource.com/cpp/RSPEC-1035 | SonarQube conducts static testing to review all code provided to ensure nothing is breaking any rules inset into the program. |

### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Naming Convention of Files | [STD-008-CPP] | Every language has keywords that should never be used otherwise it will disrupt the program. |

| **Noncompliant Code** |
| --- |
| Include\_next is specific to gcc and if ran on certain machines, can cause complier errors. |
| #include\_next "foo.h" // Noncompliant |

| **Compliant Code** |
| --- |
| For files that may be clashing with the system, you should utilize words that are native to the compiler. |
| #include "/usr/local/include/foo.h" |

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

|  |
| --- |
| **Principles(s):** Rule 1 |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Minor | Unlikely | Low | Minor | 1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| SonarQube | 9.5 | https://rules.sonarsource.com/cpp/RSPEC-3730 | SonarQube conducts static testing to review all code provided to ensure nothing is breaking any rules inset into the program. |

### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Class protection | [STD-009-CPP] | Child classes can see protected members but not private. When using a final class there is no protected members only private. |

| **Noncompliant Code** |
| --- |
| The code below utilizes a protected class instead of private class. This isn’t needed since it’s a final class. |
| class C final { protected: // Noncompliant  void fun(); }; |

| **Compliant Code** |
| --- |
| Changing protected class to private class resolves the issue. |
| class C final { private: // compliant  void fun(); }; |

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

|  |
| --- |
| **Principles(s):** Rule 2 |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Minor | Unlikely | Low | Minor | 2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| SonarQube | 9.5 | https://rules.sonarsource.com/cpp/RSPEC-2156 | SonarQube conducts static testing to review all code provided to ensure nothing is breaking any rules inset into the program. |

### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| No wasted code | [STD-010-CPP] | Wasted code takes up compile/memory space that if ever leaked could expose thigs we don’t want. This goes back to our rule about least access. |

| **Noncompliant Code** |
| --- |
| Below shows obsolete code just beyond the return statement. This code will never be accessed by the program. |
| int fun(int a) {  int i = 10;  return i + a; // Noncompliant  i++; // dead code } |

| **Compliant Code** |
| --- |
| The best practice would be to remove the obsolete code. |
| int fun(int a) {  int i = 10;  return i + a;  } |

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

|  |
| --- |
| **Principles(s):** Rule 2 |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Major | Low | Low | Major | 2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| SonarQube | 9.5 | https://rules.sonarsource.com/cpp/RSPEC-1763 | SonarQube conducts static testing to review all code provided to ensure nothing is breaking any rules inset into the program. |

## 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 | Major | Unlikely | Medium | High | 2 |
| STD-002-CPP | Blocker | Semi-Likely | Medium-High | High | 5 |
| STD-003-CPP | Major | Unlikely | Small | Major | 1 |
| STD-004-CPP | Major | Likely | High | Major | 5 |
| STD-005-CPP | Major | Likely | High | Major | 5 |
| STD-006-CPP | Low | Medium | Low | Low | 1 |
| STD-007-CPP | Critical | Unlikely | Medium | High | 3 |
| STD-008-CPP | Minor | Unlikely | Low | Minor | 1 |
| STD-009-CPP | Minor | Unlikely | Low | Minor | 2 |
| STD-010-CPP | Major | Low | Low | Major | 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 | Encryption in rest is data that is encrypted only when not in use. This is used so that if data is ever extracted for malicious use, it won’t be readable unless they know the decryption key. |
| Encryption at flight | Encryption at flight is where data is encrypted right before transmission. This method is used to ensure any data captured mid transfer will be unusable unless the appropriate decryption key is known. |
| Encryption in use | Encryption in use is a combination of the previous two. This method is where the data is encrypted all the time. This is data that you must keep secret at all costs. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication is the process in which you verify who someone is by any means that you deem necessary. By doing this, you ensure no malicious users gain access to any data they shouldn’t. |
| Authorization | Authorization comes after authentication. This is for what the user can and cannot do while they have access to the data. This is meant to curtail the use and manipulation of data by the users unless the need to. |
| Accounting | Accounting is the documentation of all the user’s activities. This helps track movement and to place blame if something goes wrong. |

**\***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 | 07/17/2022 | Initial Template | Nicholas Els |  |
| 2.0 | 08/06/2022 | Reformed Template | Nicholas Els |  |
| [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 |