**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 | [Ensure that any data that is being input into a system is error free and validated in order to prevent any vulnerabilities or cause errors within a system.] |
| 1. Heed Compiler Warnings | [indicates potential warnings within code that could perhaps lead to bigger issues or vulnerabilities even if the code still runs.] |
| 1. Architect and Design for Security Policies | [creating a security policy plan early on in the development process in order to adhere to them and make sure they are correctly followed to maintain best practices.] |
| 1. Keep It Simple | [ensure that the code is made simple and legible in order to maintain higher legibility as well as ensuring any potential fixes can be made quickly and with ease.] |
| 1. Default Deny | [to ensure maximum security deny any access or potential privileges unless otherwise required or authorized.] |
| 1. Adhere to the Principle of Least Privilege | [priority should be set on giving the least privilege required to complete any given task in order to maximize the security and minimize the threat.] |
| 1. Sanitize Data Sent to Other Systems | [ensure that any form of data being sent to another system adheres to their formatting and is valid in order to minimize causing any potential issues when the other system receives and tries to process the data.] |
| 1. Practice Defense in Depth | [the use of including different layers of security within a system in order to maximize security as if one layer fails the next one is there to prevent any further issues.] |
| 1. Use Effective Quality Assurance Techniques | [ensure that you are thoroughly testing your code with effective quality assurance techniques in order to stay on top of any potential vulnerabilities and maintain the upmost security within your system.] |
| 1. Adopt a Secure Coding Standard | [maintains the code at the highest level possible ensuring that best coding practices are followed in order to maintain proper well rounded and secure code throughout the entire development process.] |

### 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] | [data types should be declared as close to first use as possible] |

| **Noncompliant Code** |
| --- |
| [demonstrates object being declared incorrectly within a for loop unnecessarily taking up extra memory and cpu resources. ] |
| *// Inefficient implementation:*  *for (int i = 0; i < 1000000; ++i) {*  *Foo f; // ctor and dtor get called 1000000 times each*  *f.DoSomething(i);*  *}* |

| **Compliant Code** |
| --- |
| [Compliant description] |
| *Foo f; // My ctor and dtor get called once each.*  *for (int i = 0; i < 1000000; ++i) {*  *f.DoSomething(i);*  *}* |

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

| **Principles(s):** Keep it simple: by maintaining the scope of the variable more limited we increase simplicity and legibility of the code. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| [low] | [low] | [low] | [low] | [3] |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CppCheck | 2.17.1 | Cert Coding Standards | CppCheck is a static analyses tool that checks C/C++ code for bugs, errors, and dangerous coding constructs. |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | [STD-002-CPP] | [ensure numerical data values do not get distorted through data conversions such as implicit or explicit conversions.] |

| **Noncompliant Code** |
| --- |
| [this example creates loss of data through the shortening of the function] |
| *#include <limits.h>*    *void func(void) {*  *signed long int s\_a = LONG\_MAX;*  *signed char sc = (signed char)s\_a; /\* Cast eliminates warning \*/*  */\* ... \*/*  *}* |

| **Compliant Code** |
| --- |
| [validates ranges within the conversion of signed type] |
| *#include <limits.h>*    *void func(void) {*  *signed long int s\_a = LONG\_MAX;*  *signed char sc;*  *if ((s\_a < SCHAR\_MIN) || (s\_a > SCHAR\_MAX)) {*  */\* Handle error \*/*  *} else {*  *sc = (signed char)s\_a; /\* Use cast to eliminate warning \*/*  *}*  */\* ... \*/*  *}* |

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

| **Principles(s):** validate input data: reduce any potential loss in accuracy when working with num data types. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| medium | medium | low | medium | 2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CppCheck | 2.17.1 | Cert Coding Standards | CppCheck is a static analyses tool that checks C/C++ code for bugs, errors, and dangerous coding constructs. |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | [STD-003-CPP] | Ensure that character data for strings has enough storage to protect functionality. |

| **Noncompliant Code** |
| --- |
| [a buffer overflow is possible to occur when trying to write more than 12 characters for one array] |
| *#include <iostream>*    *void f() {*  *char bufOne[12];*  *char bufTwo[12];*  *std::cin.width(12);*  *std::cin >> bufOne;*  *std::cin >> bufTwo;*  *}* |

| **Compliant Code** |
| --- |
| [to make things less complex a simple std::string may help over an array] |
| *#include <iostream>*  *#include <string>*    *void f() {*  *std::string input;*  *std::string stringOne, stringTwo;*  *std::cin >> stringOne >> stringTwo;*  *}* |

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

| **Principles(s):** validate input data: ensuring you manage the strings correctly and the input data is properly managed you can prevent any possible errors. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| medium | medium | low | medium | 2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CppCheck | 2.17.1 | Cert Coding Standards | CppCheck is a static analyses tool that checks C/C++ code for bugs, errors, and dangerous coding constructs. |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | [STD-004-CPP] | Prevent any possible leaks or data losses by limiting or stopping any SQL injection attacks. |

| **Noncompliant Code** |
| --- |
| [user input here is being used in an SQL query directly which leaves you open to potential attacks.] |
| *String sqlString = "SELECT \* FROM db\_user WHERE username = '"*  *+ username +*  *"' AND password = '" + pwd + "'";*  *Statement stmt = connection.createStatement();*  *ResultSet rs = stmt.executeQuery(sqlString);* |

| **Compliant Code** |
| --- |
| [bypass by potentially using something like ASP.net] |
| *private void UseCommandObject(){*  *Dim strSQL as String sql*  *Dim SqlCommand As OleDbCommand*  *cmdSQL = New SQLDbCommand()*  *strSQL = "SELECT \* FROM tblUSER "*  *sql += " WHERE USERId = @UserId"*  *cmdSQL.CommandText = strSQL*  *cmdSQL.Parameters.Add(new SqlParameter("@UserId","Test"));*  *cmdSQL.Connection = new SqlConnection("Server=Localhost;Database=Northwind; +*  *Integrated Security=Yes")*  *cmdSQL.ExecuteNonQuery()*  *}* |

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

| **Principles(s):** sanitize data sent to other systems: by sanitizing and properly formatted to be sent or used by other systems you better protect yourself from potential attacks. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| high | medium | medium | medium | 2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CppCheck | 2.17.1 | Cert Coding Standards | CppCheck is a static analyses tool that checks C/C++ code for bugs, errors, and dangerous coding constructs. |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | [STD-005-CPP] | Buffer overflow, underflow and or memory leaks and any other common memory access errors should be prevented |

| **Noncompliant Code** |
| --- |
| [no space allocated which results in an overflow] |
| *#include <algorithm>*  *#include <vector>*    *void f() {*  *std::vector<int> v;*  *std::fill\_n(v.begin(), 10, 0x42);*  *}* |

| **Compliant Code** |
| --- |
| [the capacity of the vector is set initially] |
| *#include <algorithm>*  *#include <vector>*    *void f() {*  *std::vector<int> v(10);*  *std::fill\_n(v.begin(), 10, 0x42);*  *}* |

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

| **Principles(s):** validate input data: this ensures that any potential errors or over/underflow is prevented. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| high | high | medium | high | 1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CppCheck | 2.17.1 | Cert Coding Standards | CppCheck is a static analyses tool that checks C/C++ code for bugs, errors, and dangerous coding constructs. |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | [STD-006-CPP] | Essentially a debugging tool used to detect logical errors that can potentially cause any vulnerabilities |

| **Noncompliant Code** |
| --- |
| [this code basically only executes at runtime which makes it far more difficult to detect errors beforehand] |
| *#include <assert.h>*    *struct timer {*  *unsigned char MODE;*  *unsigned int DATA;*  *unsigned int COUNT;*  *};*    *int func(void) {*  *assert(sizeof(struct timer) == sizeof(unsigned char) + sizeof(unsigned int) + sizeof(unsigned int));*  *}* |

| **Compliant Code** |
| --- |
| [allows it to compile and detect these errors when compiling ] |
| *#include <assert.h>*    *struct timer {*  *unsigned char MODE;*  *unsigned int DATA;*  *unsigned int COUNT;*  *};*    *static\_assert(sizeof(struct timer) == sizeof(unsigned char) + sizeof(unsigned int) + sizeof(unsigned int),*  *"Structure must not have any padding");* |

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

| **Principles(s):** architect and design for security: test driven development makes it so errors can be caught before the actual code is run |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| medium | low | low | medium | 2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Google Test | 1.14.0 | Unit test automation framework | Automated unit tests for C++ code |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | [STD-000-CPP] | Management of runtime errors in order to ensure the prevention of any disruption of the program being executed as well as any possible security issues. |

| **Noncompliant Code** |
| --- |
| [does not specify what exception is thrown leading to harder time managing and handling that exception.] |
| *private void doSomething() throws Exception {*  *//...*  *}* |

| **Compliant Code** |
| --- |
| [specifies exactly what exception to throw and is expected to catch to prevent any issues] |
| *private void doSomething() throws IOException {*  *//...*  *}* |

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

| **Principles(s):** architect and design for security policies: the proper management of exceptions can prevent the disruption of the program execution and it being able to handle and manage errors that are caught. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| high | high | low | high | 1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Google Test | 1.14.0 | Unit test automation framework | Automated unit tests for C++ code |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
|  | [STD-008-CPP] | [Rationalize the standard.] |

| **Noncompliant Code** |
| --- |
| [Noncompliant description] |
| [Noncompliant code block; code should be indented using 12-point Courier New font.] |

| **Compliant Code** |
| --- |
| [Compliant description] |
| [Compliant code block; code should be indented using 12-point Courier New font.] |

**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** |
| --- | --- | --- |
|  | [STD-009-CPP] | [Rationalize the standard.] |

| **Noncompliant Code** |
| --- |
| [Noncompliant description] |
| [Noncompliant code block; code should be indented using 12-point Courier New font.] |

| **Compliant Code** |
| --- |
| [Compliant description] |
| [Compliant code block; code should be indented using 12-point Courier New font.] |

**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** |
| --- | --- | --- |
|  | [STD-010-CPP] | [Rationalize the standard.] |

| **Noncompliant Code** |
| --- |
| [Noncompliant description] |
| [Noncompliant code block; code should be indented using 12-point Courier New font.] |

| **Compliant Code** |
| --- |
| [Compliant description] |
| [Compliant code block; code should be indented using 12-point Courier New font.] |

**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 | low | low | low | low | 3 |
| STD-002-CPP | medium | medium | low | medium | 2 |
| STD-003-CPP | medium | medium | low | medium | 2 |
| STD-004-CPP | high | medium | medium | medium | 2 |
| STD-005-CPP | high | high | medium | high | 1 |
| STD-006-CPP | medium | low | low | medium | 2 |
| STD-007-CPP | high | high | low | high | 1 |

### 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 | [this is designed to maintain and keep stored data encrypted in order to prevent unwanted access or leaks without the correct key to decrypt it.] |
| Encryption in flight | [maintaining data encrypted while it is being sent one from one place to another aka while its leaving its rest state ensuring that secure network connections are established and data being transferred is encrypted and unreadable without the necessary key to decrypt it.] |
| Encryption in use | [maintains all data in use encrypted in order to maintain upmost security and uphold the security levels in all states of the data being managed while at rest, in flight or in use] |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | [the process in which the system verifies that the user is the user they claim to be either through login credentials or potentially through the use of 2FA.] |
| Authorization | [this process is mainly to do with the level of authority or privilege a user may have within the system ensuring they are allowed to access only the lowest areas or information they need in order to complete their tasks.] |
| Accounting | [this is where tracking of either transactions or requests of a user are monitored and stored in order to better keep the system running without any potential issues or to facilitate the finding of these issues if they were to arise. ] |

**\***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 |  |
| 2.0 | 06/22/2025 | Updated with completed tables | Eddy Godoy | Eddy Godoy |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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