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



# Green Pace Secure Development Policy

## Contents

[Overview 2](#_Toc52464053)

[Purpose 2](#_Toc52464054)

[Scope 2](#_Toc52464055)

[Module Three Milestone 2](#_Toc52464056)

[Ten Core Security Principles 2](#_Toc52464057)

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

[Coding Standard 1 4](#_Toc52464059)

[Coding Standard 2 5](#_Toc52464060)

[Coding Standard 3 6](#_Toc52464061)

[Coding Standard 4 7](#_Toc52464062)

[Coding Standard 5 8](#_Toc52464063)

[Coding Standard 6 9](#_Toc52464064)

[Coding Standard 7 10](#_Toc52464065)

[Coding Standard 8 11](#_Toc52464066)

[Coding Standard 9 13](#_Toc52464067)

[Coding Standard 10 14](#_Toc52464068)

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

[Project One 15](#_Toc52464070)

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

[2. Risk Assessment 15](#_Toc52464072)

[3. Automated Detection 15](#_Toc52464073)

[4. Automation 15](#_Toc52464074)

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

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

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

[Audit Controls and Management 18](#_Toc52464078)

[Enforcement 18](#_Toc52464079)

[Exceptions Process 18](#_Toc52464080)

[Distribution 19](#_Toc52464081)

[Policy Change Control 19](#_Toc52464082)

[Policy Version History 19](#_Toc52464083)

[Appendix A Lookups 19](#_Toc52464084)

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

## Overview

Software development at Green Pace requires consistent implementation of secure principles to all developed applications. Consistent approaches and methodologies must be maintained through all policies that are uniformly defined, implemented, governed, and maintained over time.

## Purpose

This policy defines the core security principles; C/C++ coding standards; authorization, authentication, and auditing standards; and data encryption standards. This article explains the differences between policy, standards, principles, and practices (guidelines and procedure): [Understanding the Hierarchy of Principles, Policies, Standards, Procedures, and Guidelines](https://www.linkedin.com/pulse/understanding-hierarchy-principles-policies-standards-wally-beddoe/).

## Scope

This document applies to all staff that create, deploy, or support custom software at Green Pace.

## Module Three Milestone

### Ten Core Security Principles

| **Principles** | Write a short paragraph explaining each of the 10 principles of security. |
| --- | --- |
| 1. ValidateInput Data | Validating input data ensures that data that is being received by the application is not malicious. This involves checking to make sure the user input is exactly what you need such as formatting, lengths, and types. When failing to do this user can exploit weak points causing major security issues. |
| 1. Heed Compiler Warnings | Compiler warnings are crucial to fix ensuring secure and reliable code. Treat Compiler warning as errors and fix them before releasing code. These warnings can be exploited and cause system crashes if not fixed. |
| 1. Architect and Design for Security Policies | Security should be incorporated into the design of the system. It shouldn’t be added later but planned for during the design phase. When embedding security polices in the design it will help build systems that are more secure. |
| 1. Keep It Simple | Simple code is important and ensures that the code is easy to understand and maintain. When things become too complex it’s hard to reduce unwanted behavior and flaws that can reveal themselves later. Simple is better and always more reliable than complex. |
| 1. Default Deny | When designing systems, access should always be denied unless allowing access. This ensures that the code is secure, and users are only given access to what is needed. This is based on the least privilege which means giving the user the least amount of permission while ensuring the system still functions. |
| 1. Adhere to the Principle of Least Privilege | PoLP ensures that the system should only be granting the minimum level of access to perform the task needed. This will reduce risk and ensure users aren’t able to exploit access they don’t have permission to. |
| 1. Sanitize Data Sent to Other Systems | Clean data to ensure SQL injections can’t be exploited. This looks at script tags and SQL commands that are added to the input in hopes of exploiting systems. Without sanitization, attackers can easily exfiltrate data. This ensures the safety of the system, and the external systems connected. |
| 1. Practice Defense in Depth | Apply multiple layers of security to ensure security. The goal is to ensure that if one security system fails another will still provide protection. This is a layered approach that will greatly increase security. There is no single solution that is secure so having multiple can enhance security. |
| 1. Use Effective Quality Assurance Techniques | Apply quality assurance techniques such as static/dynamic analysis, automated testing, and code reviews to ensure security. Doing this should help uncover unknown issues that can be fixed before the system is released. |
| 1. Adopt a Secure Coding Standard | Use code standard that will help ensure consistency as well as eliminate common vulnerabilities. These are clear rules that developers need to follow to ensure the security of the system. When being consistent with coding standards security will greatly increase. |

### 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] | Validate all input data |

| **Noncompliant Code** |
| --- |
| Input is not validated before use, risking injection or buffer overflow. |
| char buffer[10];  strcpy(buffer, input); This can overflow buffer |

| **Compliant Code** |
| --- |
| Input length and format are verified before processing any data |
| if(strlen(input) < sizeof(buffer)) {  strcpy(buffer, input);  } else {  //Some error handling based on input  } |

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

| **Principles(s):** Validate Input Data – This makes sure that the code is safe and the input that is expected is processed in the code. This will reduce overflow and injection risk. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Cppcheck | 2.10 | bufferOverflow | Will detect buffer writes that are dangerous. |

Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | [STD-002-CPP] | Simplify handling and least privilege |

| **Noncompliant Code** |
| --- |
| Privileges are to broad and code is complex |
| void process() {  runAsAdmin(); //Elevated privilege not necessary  //Complex coding logic  } |

| **Compliant Code** |
| --- |
| Least privilege applied and simple code |
| void process() {  runAsUser(); //Least privilege  //Simple coding logic  } |

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

| **Principles(s):** Keep It Simple – Simplifies logic to reduce possible errors  Principle of Least Privilege – Ensures the user only has access to the necessary permissions |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| SonarQube | 9.9 | S2438 | Will detect over complex functions and excessive privilege |

#### 

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | [STD-003-CPP] | Secure string handling and sanitization |

| **Noncompliant Code** |
| --- |
| String concatenation is unsafe leading to possible injection |
| char query[256];  sprintf(query, “SELECT \* FROM users WHERE name=’%s’”, input); |

| **Compliant Code** |
| --- |
| String handling is secure and input is sanitized |
| char query[256];  sanitize(input);  snprintf(query, sizeof(query), “SELECT \* FROM users WHERE name=’%s’”, input); |

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

| **Principles(s):** Validate Input Data – This makes sure that the code is safe and the input that is expected is processed in the code. This will reduce overflow and injection risk.  Sanitize Data Sent to Other Systems – Ensures any malicious content is removed before sent |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | Medium | High | 4 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Fortify SCA | 23.1 | Poor input validation | Detects inputs that aren’t cleaned in SQL strings |

#### Coding Standard 4

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

| **Noncompliant Code** |
| --- |
| Direct concatenation of SQL with no defense |
| executeSQL(“DELETE FROM users WHERE id=” + userInput); |

| **Compliant Code** |
| --- |
| Use of queries that are parameterized as well as layered |
| prepareSQL(“DELETE FROM users WHERE id=” + userInput);  bindParam(userInput);  executePrepared(); |

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

| **Principles(s):** Architect for Security – Protection from SQLi at design.  Sanitize Data Sent to Other Systems – Ensures any malicious content is removed before sent |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | High | High | 5 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| SonarQube | 9.9 | S3649 | Will identify SQL concatenation and will ensure parameterization |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | [STD-005-CPP] | Compiler support and memory protection |

| **Noncompliant Code** |
| --- |
| Ignoring compiler warnings can cause undefined behaviors |
| int \*ptr = nullptr;  \*ptr = 10; // Null pointer reference |

| **Compliant Code** |
| --- |
| Fix warnings and protect memory |
| int val = 10;  int \*ptr = &val;  \*ptr = 10; |

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

| **Principles(s):** Heed Compiler Warnings – This will avoid any undefined behavior  Use Effective QA Techniques – Early bug detection |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Clang-Tidy | 15 | Cppcoreguidelines-pro-bounds-arithmetic | Will catch any null or unsafe pointer being used |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | [STD-006-CPP] | Assertion handling and default deny |

| **Noncompliant Code** |
| --- |
| Assertions disabled, allowing unauthorized access |
| assert(user != nullptr);  grantAccess(user); |

| **Compliant Code** |
| --- |
| Enforcing access control with checks. Only use assertions during development |
| if (user == nullptr) {  denyAccess();  return;  }  Assert(user != nullptr); \\ Used for testing  grantAccess(user); |

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

| **Principles(s):** Default Deny – Will only allow access when verified  Adopt a Secure Coding Standard – Using the correct assertion is important |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Cppcheck | 2.10 | assertWithSideEffect | Will detect assertions that may act differently when released |

#### 

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | [STD-007-CPP] | Exception handling and defense in layers |

| **Noncompliant Code** |
| --- |
| No exception handling, can cause crashes |
| void func() {  //No try or catch  riskyOperation();  } |

| **Compliant Code** |
| --- |
| Proper exception handling |
| void func() {  try {  riskyOperation();  }  catch() {  handleError();  }  } |

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

| **Principles(s):** Defense in Depth – Adds additional layers to fall back on  Use Effective QA Techniques – Will ensure error etiquette handling |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | Medium | High | 4 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Coverity | 2024.03 | UNCAUGHT\_EXCEPT | Will detect unhandled exception paths |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Resource Allocation | [STD-008-CPP] | Secure resource allocation |

| **Noncompliant Code** |
| --- |
| Resource allocated without checking |
| char \*ptr = new char[size]; |

| **Compliant Code** |
| --- |
| Checked resource allocation |
| Char \*ptr = new char[size];  if(ptr == nullptr) {  // Handle error  } |

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

| **Principles(s):** Keep It Simple – Will prevent unexpected behaviors  Adopt a Secure Coding Standard – Ensure resource allocation is safe |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Valgrind | 3.21 | Memcheck | Will detect improper allocation and memory leaks |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Error Checking | [STD-009-CPP] | Performance and error checking |

| **Noncompliant Code** |
| --- |
| No boundary or error checking can cause errors |
| for(int i = 0; i <= size; i++) {  process(data[i]);  } |

| **Compliant Code** |
| --- |
| Proper bounds and error checking |
| for(int i = 0; i <= size; i++) {  if(data[i] != nullptr) {  process(data[i]);  }  } |

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

| **Principles(s):** Use Effective QA Techniques – Will help reduce crash risk  Adopt a Secure Coding Standard – Ensures safer code |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Clang Static Analyzer | 16 | Core.CallandMessage | Will detect bounds violations and unvalidated pointers |

#### 

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Error Management | [STD-010-CPP] | Simple and secure error management |

| **Noncompliant Code** |
| --- |
| Complex and error prone handling |
| if(error) {  logError();  // Complex recovery code  } |

| **Compliant Code** |
| --- |
| Simple and secure error handling |
| if(error) {  logError();  return FAILURE;  } |

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

| **Principles(s):** Keep It Simple – Will Simplify response paths  Adopt a Secure Coding Standard – Best practice for error handling |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Possible | Low | Low | 2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| SonarQube | 9.9 | S112 | Will detect conditions that are too complex or vague |

### 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.

The first thing is plan which will include defining the security standards and policies to ensure a safer program. Planning is integral to getting the rest of the process correct. Next step is design where the software will take the plan from before and start to implement more such as the way it will work and the test that will be done. Build is where the coding will start but will also include testing during the process to ensure everything is caught in this stage. Next is to verify where there will be more scanning and dynamic analysis to find any issues that weren’t caught during the build stage. Transition is where the program will be released with more security as well as testing to ensure secure code. Monitoring is very important and will involve checking logs and other analytics to ensure nothing is happening that is unexpected. The respond will be if an issue does arise how and what do we do to fix the issue. This could be shutting it down and fixing the problem with more testing before release. Maintain is once an attack happens and we respond we need to ensure it is still secure and continue to find and fix issues before anything else happens.

### 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 |
| STD-002-CPP | Medium | Likely | Medium | Medium | 3 |
| STD-003-CPP | High | Likely | Medium | High | 4 |
| STD-004-CPP | High | Likely | High | High | 5 |
| STD-005-CPP | High | Unlikely | Low | Medium | 3 |
| STD-006-CPP | Medium | Unlikely | Low | Medium | 2 |
| STD-007-CPP | High | Likely | Medium | High | 4 |
| STD-008-CPP | Medium | Likely | Medium | Medium | 3 |
| STD-009-CPP | Medium | Possible | Low | Medium | 3 |
| STD-010-CPP | Low | Possible | Low | Low | 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 at rest | The data store on any sort of disk should be secure by using encryption. This will ensure anything such as stolen data cannot be recovered. This will be everything from databases to files. This should be used to secure data and ensure that even if the data is taken its not able to be used. |
| Encryption in flight | A layer of TLS should be used to ensure that the connection between the client and server is secure. TLS 1.2 is widely used and considered to be secure. The reason behind this is to ensure attacks like man in the middle aren’t able to grab information transmitted. |
| Encryption in use | Memory encryption must be used to ensure data that is being processed is not accessible to users that shouldn’t have it. TME is Intel’s version of Total Memory Encryption which encrypts all data passing to and from the CPU to ensure security. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Users must use MFA, known as multifactor authentication before accessing any systems to ensure the user is truly allowed access. During this all of the login’s will be logged to ensure the MFA has not been compromised. |
| Authorization | Role-based access control will be required. This ensures that users only have access to the data they are given permission to access. This falls inline with least privilege by giving users the bare minimum required access. |
| Accounting | All of the users actions will be logged. This is logins, access, and changes to data. This will be stored for 200 days or less. This will ensure that everything from the last 200 days is accessible and audited to ensure security. |

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

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

### Map the Principles

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

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

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

|  |  |  |
| --- | --- | --- |
| STD-001-CPP | 1,6 | Will validate the input as well as applies least privilege to ensure security. |
| STD-002-CPP | 4,6 | Ensures simplicity as well as permissions that aren’t necessary. |
| STD-003-CPP | 1,7 | Will validate the input as well as sanitizing the input to prevent injection. |
| STD-004-CPP | 3,7 | Ensures a secure design as well as sanitizing to prevent injection. |
| STD-005-CPP | 2,9 | Ensures compiler warning are fixed as well as improving memory safety using QA practices. |
| STD-006-CPP | 5,10 | Uses deny-by-default as well as coding practices that are reliable and safe. |
| STD-007-CPP | 8,9 | Ensures defense in depth as well as improves exception handling using QA practices. |
| STD-008-CPP | 4,10 | Ensures allocation checks as well as standard practices. |
| STD-009-CPP | 9,10 | Ensures error checks to prevent crashes as well as following secure standards. |
| STD-010-CPP | 4,10 | Ensures simple and secure error handling by using the best coding practices. |

The only item you must complete beyond this point is the Policy Version History table.

## Audit Controls and Management

Every software development effort must be able to provide evidence of compliance for each software deployed into any Green Pace managed environment.

Evidence will include the following:

* Code compliance to standards
* Well-documented access-control strategies, with sampled evidence of compliance
* Well-documented data-control standards defining the expected security posture of data at rest, in flight, and in use
* Historical evidence of sustained practice (emails, logs, audits, meeting notes)

## Enforcement

The office of the chief information security officer (OCISO) will enforce awareness and compliance of this policy, producing reports for the risk management committee (RMC) to review monthly. Every system deployed in any environment operated by Green Pace is expected to be in compliance with this policy at all times.

Staff members, consultants, or employees found in violation of this policy will be subject to disciplinary action, up to and including termination.

## Exceptions Process

Any exception to the standards in this policy must be requested in writing with the following information:

* Business or technical rationale
* Risk impact analysis
* Risk mitigation analysis
* Plan to come into compliance
* Date for when the plan to come into compliance will be completed

Approval for any exception must be granted by chief information officer (CIO) and the chief information security officer (CISO) or their appointed delegates of officer level.

Exceptions will remain on file with the office of the CISO, which will administer and govern compliance.

## Distribution

This policy is to be distributed to all Green Pace IT staff annually. All IT staff will need to certify acceptance and awareness of this policy annually.

## Policy Change Control

This policy will be automatically reviewed annually, no later than 365 days from the last revision date. Further, it will be reviewed in response to regulatory or compliance changes, and on demand as determined by the OCISO.

## Policy Version History

| Version | Date | Description | Edited By | Approved By |
| --- | --- | --- | --- | --- |
| 1.0 | 08/05/2020 | Initial Template | David Buksbaum |  |
| 1.2 | 12/06/2025 | Security Policy | Brandon Ayers | [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 |