**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 | All user input must be checked for type, length, format, and range. This prevents injection attacks and data corruption caused by malformed or malicious input. |
| 1. Heed Compiler Warnings | Compiler and static analysis warnings highlight potential flaws in the code. Addressing these helps prevent security vulnerabilities before deployment. |
| 1. Architect and Design for Security Policies | Security must be integrated during the design phase, not as an afterthought. This includes threat modeling, data flow analysis, and adherence to secure design patterns. |
| 1. Keep It Simple | Simplicity reduces the likelihood of mistakes. Code and systems that are easy to understand are also easier to audit and maintain securely. |
| 1. Default Deny | Access should be denied unless explicitly granted. This minimizes the risk of unauthorized access due to unanticipated conditions. |
| 1. Adhere to the Principle of Least Privilege | Programs should run with the least amount of privilege required. This limits the impact of vulnerabilities by reducing the level of access an attacker gains. |
| 1. Sanitize Data Sent to Other Systems | Output data must be cleansed to prevent injection attacks when interfacing with other systems (e.g., databases, browsers, or shell commands). |
| 1. Practice Defense in Depth | Multiple layers of security should be implemented. If one layer is breached, others can still provide protection. |
| 1. Use Effective Quality Assurance Techniques | Regular testing, code reviews, and audits help ensure that security flaws are detected and resolved early. |
| 1. Adopt a Secure Coding Standard | Following established coding guidelines such as SEI CERT helps ensure consistent, secure practices across the development team. |

### 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** | **Use appropriate data types for variables** |
| --- | --- | --- |
| **Data Type** | STD-001-CPP | Using data types that are too small or inappropriate can cause overflows or truncation, leading to vulnerabilities or incorrect program behavior. |

| **Noncompliant Code** |
| --- |
| Assigning a value beyond the range of a small integer type. |
| short val = 40000; // Exceeds range of 'short' |

| **Compliant Code** |
| --- |
| Use a data type large enough to hold the expected values. |
| int val = 40000; |

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

| **Principles(s):** Validate Input Data, Adopt a Secure Coding Standard |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Clang-Tidy | 14.0 | Misc-integer-sign | Detects implicit narrowing conversions and suggests safer alternatives |
| [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** | **Validate critical data values** |
| --- | --- | --- |
| **Data Value** | STD-002-CPP | Unchecked values can cause unexpected results, logic errors, or even vulnerabilities if assumptions are violated. |

| **Noncompliant Code** |
| --- |
| Not checking array index bounds. |
| int arr[5];  arr[5] = 10; // Out-of-bounds access |

| **Compliant Code** |
| --- |
| Validate index before accessing array. |
| if (index >= 0 && index < 5) {  arr[index] = 10;  } |

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

| **Principles(s):** Validate Input Data, Keep It Simple |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | High | Low | [Insert text.] | 5 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Coverity | 2023.1 | array\_bounds | Detects out-of-bounds accesses |
| [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** | **Avoid buffer overflows in string operations** |
| --- | --- | --- |
| **String Correctness** | STD-003-CPP | Improper use of string functions can result in buffer overflows and security vulnerabilities. |

| **Noncompliant Code** |
| --- |
| Using strcpy without size validation. |
| char buffer[10];  strcpy(buffer, input); // Potential overflow |

| **Compliant Code** |
| --- |
| Use strncpy with explicit size limit. |
| char buffer[10];  strncpy(buffer, input, sizeof(buffer) - 1);  buffer[9] = '\0'; |

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

| **Principles(s):** Sanitize Data Sent to Other Systems, Adopt a Secure Coding Standard |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | High | Medium | [Insert text.] | 5 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Cppcheck | 2.10 | bufferOverflow | Detects unsafe string manipulations |
| [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** | **Use parameterized queries** |
| --- | --- | --- |
| **SQL Injection** | STD-004-CPP | Concatenating SQL strings with user input can result in SQL injection attacks. |

| **Noncompliant Code** |
| --- |
| Concatenating user input directly into query. |
| std::string query = "SELECT \* FROM users WHERE id = " + userInput; |

| **Compliant Code** |
| --- |
| Use parameterized queries. |
| stmt->prepare("SELECT \* FROM users WHERE id = ?");  stmt->bind(1, userInput); |

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

| **Principles(s):** Sanitize Data Sent to Other Systems, Default Deny |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | High | Medium | [Insert text.] | 5 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| SonarQube | 9.9 | sql-injection | Identifies raw SQL injection points |
| [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** | **Free dynamically allocated memory properly** |
| --- | --- | --- |
| **Memory Protection** | STD-005-CPP | Failing to release memory or double-freeing memory can lead to memory leaks and corruption. |

| **Noncompliant Code** |
| --- |
| Forgetting to free allocated memory. |
| char\* buffer = new char[10];  // Forgot: delete[] buffer; |

| **Compliant Code** |
| --- |
| Use smart pointers or ensure proper deallocation. |
| std::unique\_ptr<char[]> buffer(new char[10]); |

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

| **Principles(s):** Practice Defense in Depth, Use Effective QA Techniques |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Medium | Medium | Low | [Insert text.] | 3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Valgrind | 3.21 | memcheck | Identifies memory leaks and use-after-free errors |
| [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** | **Do not use assertions for runtime error checking** |
| --- | --- | --- |
| **Assertions** | STD-006-CPP | Assertions can be disabled in production. Critical runtime checks should always be active. |

| **Noncompliant Code** |
| --- |
| Using assert for input validation. |
| assert(input != nullptr); |

| **Compliant Code** |
| --- |
| Use explicit error handling. |
| if (input == nullptr) {  throw std::invalid\_argument("Null input");  } |

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

| **Principles(s):** Use Effective QA Techniques, Adopt a Secure Coding Standard |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Medium | Medium | Low | [Insert text.] | 3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Clang Static Analyzer | 14.0 | security.insecureAPI.assert | Flags assert misuse |
| [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** | **Catch exceptions by reference** |
| --- | --- | --- |
| **Exceptions** | STD-007-CPP | Catching exceptions by value can lead to object slicing. Always catch by reference. |

| **Noncompliant Code** |
| --- |
| Catching by value. |
| try {  // code  } catch (std::exception e) {  // handle  } |

| **Compliant Code** |
| --- |
| Catching by reference. |
| try {  // code  } catch (const std::exception& e) {  // handle  } |

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

| **Principles(s):** Keep It Simple, Use Effective QA Techniques |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Medium | Medium | Low | [Insert text.] | 3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Clang-Tidy | 14.0 | modernize-avoid-c-arrays | Warns on incorrect exception practices |
| [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** | **Validate input at all program boundaries** |
| --- | --- | --- |
| **Input Validation** | STD-008-CPP | Input from external sources should always be validated to ensure it meets expected formats and constraints before use. |

| **Noncompliant Code** |
| --- |
| Using user input without validation |
| int age = std::stoi(userInput); |

| **Compliant Code** |
| --- |
| Validate input before conversion |
| if (isDigits(userInput)) {  int age = std::stoi(userInput);  } |

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

| **Principles(s):** Validate Input Data, Default Deny |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Medium | Low | [Insert text.] | 4 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| SonarQube | 9.9 | cxx:S5430 | Detects missing input validation |
| [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** | **Use approved cryptographic libraries and algorithms** |
| --- | --- | --- |
| **Cryptographic API Usage** | STD-009-CPP | Custom or outdated cryptography can introduce serious vulnerabilities. Only vetted libraries should be used. |

| **Noncompliant Code** |
| --- |
| Using homegrown encryption |
| std::string encrypt(std::string input) {  return input + "XYZ"; // Not real encryption  } |

| **Compliant Code** |
| --- |
| Using OpenSSL AES API |
| EVP\_EncryptInit\_ex(ctx, EVP\_aes\_256\_cbc(), NULL, key, iv); |

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

| **Principles(s):** Practice Defense in Depth, Adopt a Secure Coding Standard |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Medium | Medium | [Insert text.] | 4 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Fortify | 23.1 | InsecureCrypto | Detects weak crypto implementations |
| [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** | **Avoid logging sensitive information** |
| --- | --- | --- |
| **Logging Practices** | STD-010-CPP | Logging passwords, tokens, or personal data can lead to data breaches if logs are compromised. |

| **Noncompliant Code** |
| --- |
| Logging sensitive data |
| log("User password: " + password); |

| **Compliant Code** |
| --- |
| Mask or omit sensitive data |
| log("User login attempt detected."); |

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

| **Principles(s):** Adhere to Least Privilege, Use Effective QA Techniques |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Medium | Low | [Insert text.] | 4 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Brakeman | 5.3 | Sensitive-data | Detects logging of sensitive information |
| [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.

The DevSecOps diagram outlines a continuous integration and deployment pipeline where security is integrated at every phase of the software development lifecycle. To enforce the standards defined in this policy, Green Pace’s DevOps process should be modified at three key stages: Code, Build, and Test. During the Code stage, automated static code analysis tools like Cppcheck or SonarQube should be integrated into version control hooks to detect violations of secure coding standards. In the Build stage, pipeline scripts should include security scans and fail builds when critical vulnerabilities are found. In the Test phase, automated testing should include unit tests for edge cases and security-specific test cases like input validation and boundary checks. By embedding security checks across these phases, the DevOps pipeline transforms into a DevSecOps pipeline that ensures continuous compliance and reduces manual oversight.

### 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 | High | High | Low | High | 5 |
| STD-003-CPP | High | High | Medium | High | 5 |
| STD-004-CPP | High | High | Medium | High | 5 |
| STD-005-CPP | Medium | Medium | Low | Medium | 3 |
| STD-006-CPP | Medium | Medium | Low | Medium | 3 |
| STD-007-CPP | Medium | Medium | Low | Medium | 3 |
| STD-008-CPP | High | Medium | Low | High | 4 |
| STD-009-CPP | High | Medium | Medium | High | 4 |
| STD-010-CPP | High | Medium | Low | High | 4 |

### 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 | Data stored on disk must be encrypted using strong algorithms such as AES-256. This ensures that if physical media is lost or stolen, the data remains protected. All backups, databases, and logs must be encrypted at rest. |
| Encryption in flight | All data transmitted between systems, including APIs and user interfaces, must be encrypted using TLS 1.2 or higher. This prevents interception and tampering of data during transmission. |
| Encryption in use | Where feasible, data in memory should be protected using techniques such as runtime encryption or secure enclaves. This helps safeguard sensitive data during active use and computation. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication verifies user identity through secure mechanisms such as multi-factor authentication (MFA). Every system must require user logins with strong credentials and maintain login session security. |
| Authorization | Authorization determines what actions an authenticated user may perform. Access control policies must restrict users to only the data and functions necessary for their role (principle of least privilege). |
| Accounting | Accounting ensures that all actions by users are logged and auditable. System logs should track login attempts, database changes, user access to files, and role or permission changes. These logs must be reviewed regularly to detect and investigate anomalies. |

**\***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 |  |
| 1.1 | 06/11/2025 | Revised Template | Britian Holcomb |  |
| [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 |