**Green Pace Developer: Security Policy Guide**



# 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 | Check user input to make sure it's safe and doesn't contain any malicious code. |
| 1. Heed Compiler Warnings | Pay attention to warnings from the compiler and fix any potential security issues. |
| 1. Architect and Design for Security Policies | Build security into the system from the start to prevent problems. |
| 1. Keep It Simple | Avoid making code too complicated as this can lead to security issues. |
| 1. Default Deny | Only allow access to a system or resource if the user is specifically allowed. |
| 1. Adhere to the Principle of Least Privilege | Only give users the access they need to do their job. |
| 1. Sanitize Data Sent to Other Systems | Remove or change sensitive information before sending it to other systems. |
| 1. Practice Defense in Depth | Use multiple layers of security to protect a system or resource. |
| 1. Use Effective Quality Assurance Techniques | Test and review code to make sure it's secure. |
| 1. Adopt a Secure Coding Standard | Follow guidelines and best practices to write secure code. |

### 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-GEN | Using the correct data type makes sure that variables are used correctly which reduces the risk of type related errors and improving code readability |

| **Noncompliant Code** |
| --- |
| Assigning a string to an integer variable. |
| // Noncompliant code  int x = "hello"; // Error: Cannot implicitly convert type 'string' to 'int'. |

| **Compliant Code** |
| --- |
| Assigning an integer to an integer variable. |
| // Compliant code  int x = 5; // Correctly assigning an integer to an integer variable. |

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

| **Principles(s):** *Adopt a Secure Coding Standard*: Enforces proper data type usage. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Clang Static Analyzer | Latest | core.CallAndMessage | Detects incorrect data type usage and type mismatches. |
| SonarQube | 9.9 LTS | cpp:S100 | Flags poor variable usage and unsafe type casting. |
| PVS-Studio | 7.24 | V104 | Detects suspicious implicit type conversions. |
| **Cppcheck** | 2.13 | typeError | Detects unsafe conversions and type misuse. |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | STD-002-GEN | Using correct data values make sure that variables are used correctly which reduces the risk of type related errors and improving code readability. |

| **Noncompliant Code** |
| --- |
| Assigning a null value to a non-nullable variable. |
| // Noncompliant code  int x = null; // Error: Cannot assign null to a non-nullable reference. |

| **Compliant Code** |
| --- |
| Assigning a valid value to a non nullable variable. |
| // Compliant code  int x = 5; // Correctly assigning a valid value to a non-nullable variable. |

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

| **Principles(s):** Validate Input Data: Prevents null assignment to non-nullable types. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| SonarQube | 9.9 | Cpp:S2095 | Identifies invalid/null values assigned to variables. |
| Coverity | 2023.3 | FORWARD\_NULL | Detects potential null assignments and missing checks. |
| PVS-Studio | 7.24 | V595 | Flags potential null dereferences. |
| CodeQL | Latest | cpp/implicit-null-check | Analyzes for invalid assumptions about data values. |

#### 

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | STD-003-GEN | Coding standards that are well defined enhance code readability which reduces errors and improves collaboration. |

| **Noncompliant Code** |
| --- |
| Using string concatenation instead of interpolation. |
| // Noncompliant code  string x = "Hello " + name; // Concatenation instead of interpolation. |

| **Compliant Code** |
| --- |
| Using string interpolation. |
| // Compliant code  string x = $"Hello {name}"; // Interpolation. |

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

| **Principles(s):** Keep It Simple: Reduces string concatenation error |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeQL | Latest | cpp/useless-string-operation | Flags improper or inefficient string operations. |
| SonarQube | 9.9 | cpp:S1199 | Checks for readability and improper string manipulations. |
| PVS-Studio | 7.24 | V1051 | Detects inefficient or redundant string operations. |
| Clang-Tidy | 17 | readability-string-compare | Flags poor string formatting and usage. |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | STD-004-SQL | Using parameterized queries prevents SQL injection attacks which improves code security. |

| **Noncompliant Code** |
| --- |
| Using string concatenation to build a SQL query. |
| // Noncompliant code  string query = "SELECT \* FROM users WHERE name = '" + name + "'"; // String concatenation. |

| **Compliant Code** |
| --- |
| Using parameterized queries. |
| // Compliant code  string query = "SELECT \* FROM users WHERE name = @name"; // Parameterized query.  SqlCommand cmd = new SqlCommand(query, connection);  cmd.Parameters.AddWithValue("@name", name); |

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

| **Principles(s):** Sanitize Data Sent to Other Systems: Protects against injection attacks. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | High | Medium | Critical | 5 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Fortify SCA | 22.1 | SQL Injection | Detects unparameterized SQL queries. |
| Checkmarx CxSAST | Latest | SQL Injection category | Scans for vulnerable SQL query construction. |
| SonarQube | 9.9 | sql-injection | Identifies potential injection points |
| CodeQL | Latest | cpp/sql-injection | Finds tainted data in SQL strings. |

#### 

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | STD-005-GEN | Using memory protection mechanisms prevents memory related errors and improves code security. |

| **Noncompliant Code** |
| --- |
| Not disposing of a disposable object. |
| // Noncompliant code  FileStream fs = new FileStream("file.txt", FileMode.Open); // Not disposed. |

| **Compliant Code** |
| --- |
| Using a using statement to ensure disposal. |
| // Compliant code  using (FileStream fs = new FileStream("file.txt", FileMode.Open)) { ... } // Disposed. |

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

| **Principles(s):** Practice Defense in Depth: Prevents potential memory leaks. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Valgrind | 3.21.0 | Memcheck | Detects leaks and misuse of memory. |
| AddressSanitizer | Clang integrated | Runtime instrumentation | Identifies out-of-bounds and use-after-free errors. |
| PVS-Studio | 7.24 | V507 | Detects unfreed resources or mishandled memory. |
| Cppcheck | 2.13 | Memleak | Reports memory that is not properly freed. |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | STD-006-GEN | Using assertions makes sure that code is correct and improves code readability. |

| **Noncompliant Code** |
| --- |
| Not using assertions to validate assumptions. |
| // Noncompliant code  int x = 5; // No assertion. |

| **Compliant Code** |
| --- |
| Using assertions to validate assumptions. |
| // Compliant code  int x = 5; Debug.Assert(x > 0); // Assertion. |

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

| **Principles(s):** Keep It Simple: Helps verify expected code logic. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Coverity | 2023.3 | USE\_AFTER\_FREE | Detects missing or incorrect assertion usage. |
| PVS-Studio | 7.24 | V609 | Checks for ineffective or always true assertions. |
| Clang Static Analyzer | Latest | Core.NullDerefrence | Helps in verifying logic with null assumptions. |
| Cppcheck | 2.13 | assertWithSideEffect | Flags misuse of assertions. |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | STD-007-GEN | Using exceptions correctly improves code readability and error handling. |

| **Noncompliant Code** |
| --- |
| Not catching exceptions. |
| // Noncompliant code  try { ... } // No exception handling. |

| **Compliant Code** |
| --- |
| Catching specific exceptions. |
| // Compliant code  try { ... } catch (FileNotFoundException ex) { HandleException(ex); } // Specific exception handling. |

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

| **Principles(s):** Use Effective Quality Assurance Techniques: Make certain resilience. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| PVS-Studio | 7.24 | V1024 | Flags unsafe or unhandled exceptions. |
| Clang-Tidy | 17 | Modernize-use-override | Make sure proper use of exception safety and patterns |
| SonarQube | 9.9 | Cpp:S2221 | Detects broad catch clauses or swallowing exceptions. |
| CodeQL | Latest | Cpp/exception-handling | Checks for poor or risky exception practices. |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Code Documentation | STD-008-GEN | Using code documentation improves code readability and maintainability. |

| **Noncompliant Code** |
| --- |
| Not using comments to explain code. |
| // Noncompliant code  public void DoSomething() { ... } // No comments. |

| **Compliant Code** |
| --- |
| Using comments to explain code. |
| // Compliant code  /// <summary>This method does something.</summary>  public void DoSomething() { ... } // XML comments. |

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

| **Principles(s):** Keep It Simple: Clarifies developer intentions. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Doxygen | 1.9.8 | Built in parsing | Make sure method documentation exists. |
| Sphinx | 7.0 | Autodoc | Parses and flags undocumented components. |
| ClangDoc | 17 | C++ documentation | Extract and validate sources documentation. |
| SonarQube | 9.9 | Cpp:S107 | Flags lack of comments and excessive complexity. |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Input Validation | STD-009-GEN | Using input validation improves code security and prevents errors. |

| **Noncompliant Code** |
| --- |
| Not validating user input. |
| // Noncompliant code  string name = Console.ReadLine(); // No validation. |

| **Compliant Code** |
| --- |
| Validating user input. |
| // Compliant code  string name = Console.ReadLine(); if (!string.IsNullOrEmpty(name)) { ... } // Validation. |

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

| **Principles(s):** Validate Input Data: Makes sure all inputs are safe. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | High | Low | Critical | 5 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Checkmarx CxSAST | Latest | Input Validation category | Flags missing or unsafe input checks |
| Veracode Static Analysis | Latest | Improper input validation | Identifies unsafe or unchecked input sources |
| Fortify SCA | 22.1 | Input validation | Detects risky handling of user input. |
| CodeQL | Latest | Cpp/unvalidated-input | Analyzes input flows for security flaws |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Logging Practices | STD-010-GEN | Using logging practices improves code maintainability and debugging. |

| **Noncompliant Code** |
| --- |
| Logging important events. |
| // Noncompliant code  try { ... } catch (Exception ex) { } // No logging. |

| **Compliant Code** |
| --- |
| Handling errors. |
| // Compliant code  try { ... } catch (Exception ex) { logger.LogError("Error occurred", ex); } // Logging. |

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

| **Principles(s):** Practice Defense in Depth: Supports auditing and diagnostics. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| SonarQube | 9.9 | Cpp:S1135 | Detects missing logging format |
| ELK Stack | SaaS 8.0 | Custom alerting on log anomalies | Centralized logging monitoring and validation. |
| Semgrep | Latest | Logging ruleset | Detects improper logging or missing error handlers |
| Splunk | Cloud 9.0 | Custom rules | Identifies logging gaps and monitoring runtime anomalies. |

### 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.

At Green Pace, automation plays a important role in making sure developers follow security standards without slowing down the software development process. From the very beginning of a project, teams plan for security by identifying risks and secure coding practices. While writing code, tools such as SonarLint and Clang-Tidy help developers catch and fix issues directly within their coding environments. During the build stage, automated systems like GitHub Actions or Jenkins run security checks. If a critical issue is found, the build process is stopped until it is fixed. Automated tests are also run to check that the application behaves securely and is protected against common attacks.

Before releasing software, additional security checks make sure only safe and approved code is deployed. When the software is deployed to production, tools scan the setup to make sure that permissions, configurations, and environments are secure. Once the system is live, monitoring tools like ELK Stack or Splunk track system activity, detect unusual behavior, and alert the team if something suspicious 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-GEN | Medium | Medium | Low | Medium | 3 |
| STD-002-GEN | Medium | High | Low | High | 4 |
| STD-003-GEN | Low | Medium | Low | Low | 2 |
| STD-004-SQL | High | High | Medium | Critical | 5 |
| STD-005-GEN | High | Medium | Medium | High | 4 |
| STD-006-GEN | Medium | Medium | Low | Medium | 3 |
| STD-007-GEN | High | Medium | Medium | High | 4 |
| STD-008-GEN | Low | Medium | Low | Medium | 2 |
| STD-009-GEN | High | High | Low | Critical | 5 |
| STD-010-GEN | Medium | High | 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 | When data is saved on a computer or server, it must be locked using strong encryption like AES-256. This keeps the data safe if someone gets access to the storage. |
| Encryption in flight | When data is sent between devices or systems like from a user to a website, it must be sent over secure channels like HTTPS to stop others from spying on it. |
| Encryption in use | When data is being used or processed in memory like during calculations, extra steps should be taken to protect it if it’s very sensitive, such as using secure memory spaces or masking the data. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | All users must log in securely, using something strong like a password and a phone app such multi-factor authentication. Systems should use trusted methods like OAuth to confirm identity. |
| Authorization | Once logged in, users should only see or do what they’re allowed to. Everyone gets access based on their role or job. No extra permissions should be given. |
| Accounting | Accounting tracks user actions such as logins, file changes, and permission changes. These records or logs help trace problems and prove that security is being followed. |

**\***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 logsThe 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 | 03/22/2025 | Module Three Milestone Assignment | Mohammed Khan |  |
| 1.2 | 04/08/2024 | Module Six Milestone Assignment | Mohammed Khan |  |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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