**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 | Data must be validated from all untrusted sources to ensure proper input. This eliminates the risk of most security vulnerabilities, such as SQL injection. All input should be validated from any external data sources. |
| 1. Heed Compiler Warnings | Make sure to pay attention to all compiler warnings and eliminate those warnings whenever necessary. The code should be compiled using the highest warning level possible for the compiler being used. |
| 1. Architect and Design for Security Policies | Software should be designed to implement security policies. The software should be developed with security policies in mind to allow the program to enforce those policies however necessary. |
| 1. Keep It Simple | Avoid any complex designs, as this increases the likelihood of a vulnerability. Keep the design as simple as possible to avoid errors leading to a vulnerability. This also makes it easier for the code to be modified and understood by other developers if needed. |
| 1. Default Deny | When using access decisions or authorizations, base the accept/denial of these on permissions, rather than exclusions. By default, access should be denied unless certain conditions are met to allow access. |
| 1. Adhere to the Principle of Least Privilege | Every process should be executed with the least set of privileges necessary to perform the process. If elevated permissions are used, those permissions should be held for a certain minimum time to reduce the chance of an attacker attempting to breach the system using elevated privileges. |
| 1. Sanitize Data Sent to Other Systems | Any data being sent to a complex subsystem should be sanitized beforehand. The complex subsystem does not understand the context of the call, so it is the calling process’s job to sanitize the data before sending it to the subsystem. |
| 1. Practice Defense in Depth | Using multiple defense strategies decreases the risk of an attacker finding a vulnerability and breaching the system. Using multiple defense layers also reduces the consequences if an attacker does successfully breach the system because it will be caught by the next layer. |
| 1. Use Effective Quality Assurance Techniques | Using good quality assurance techniques increases the chances of finding any vulnerabilities in the code. Using techniques such as Fuzz testing, penetration testing, and source code audits will lead to a more secure program. It would also be beneficial to use independent and external reviews. |
| 1. Adopt a Secure Coding Standard | Develop your own or use a secure coding standard to keep your program consistent and uniform so that it is more easily understood. |

### 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** | **Data Type Coding Standard** |
| --- | --- | --- |
| **Data Type** | [STD-INT32-C] | Ensure that operations on signed integers do not result in overflow |

| **Noncompliant Code** |
| --- |
| In this code, the signed integer will result in an overflow during the addition of the signed operands. |
| void func(signed int si\_a, signed int si\_b) {  signed int sum = si\_a + si\_b;  /\* ... \*/  } |

| **Compliant Code** |
| --- |
| The solution prevents the addition operation from overflowing and handles any overflow. |
| #include <limits.h>    void f(signed int si\_a, signed int si\_b) {  signed int sum;  if (((si\_b > 0) && (si\_a > (INT\_MAX - si\_b))) ||  ((si\_b < 0) && (si\_a < (INT\_MIN - si\_b)))) {  /\* Handle error \*/  } else {  sum = si\_a + si\_b;  }  /\* ... \*/  } |

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

| **Principles(s):**  **Validate Input Data**: Make sure that input values are validated to ensure no overflows occur.  **Heed Compiler Warnings**: Check the compiler warnings for any data type errors.  **Use Effective Quality Assurance Techniques:** Use testing methods to check the code for any possible overflow vulnerabilities. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | High | P9 | L2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152428) | 22.04 | **integer-overflow** | Fully checked |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 7.2p0 | **ALLOC.SIZE.ADDOFLOW** **ALLOC.SIZE.IOFLOW** **ALLOC.SIZE.MULOFLOW** **ALLOC.SIZE.SUBUFLOW** **MISC.MEM.SIZE.ADDOFLOW** **MISC.MEM.SIZE.BAD** **MISC.MEM.SIZE.MULOFLOW** **MISC.MEM.SIZE.SUBUFLOW** | Addition overflow of allocation size Integer overflow of allocation size Multiplication overflow of allocation size Subtraction underflow of allocation size Addition overflow of size Unreasonable size argument Multiplication overflow of size Subtraction underflow of size |
| [Coverity](https://wiki.sei.cmu.edu/confluence/display/c/Coverity) | 2017.07 | **TAINTED\_SCALAR**  **BAD\_SHIFT** | Implemented |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/c/Helix+QAC) | 2022.4 | **C2800, C2860**  **C++2800, C++2860**  **DF2801, DF2802, DF2803, DF2861, DF2862, DF2863** |  |
| [Klocwork](https://wiki.sei.cmu.edu/confluence/display/c/Klocwork) | 2022.4 | **NUM.OVERFLOW** **CWARN.NOEFFECT.OUTOFRANGE** **NUM.OVERFLOW.DF** |  |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/c/LDRA) | 9.7.1 | **493 S, 494 S** | Partially implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/c/Parasoft) | 2022.2 | **CERT\_C-INT32-a** **CERT\_C-INT32-b** **CERT\_C-INT32-c** | Avoid integer overflows Integer overflow or underflow in constant expression in '+', '-', '\*' operator Integer overflow or underflow in constant expression in '<<' operator |
| [Parasoft Insure++](https://wiki.sei.cmu.edu/confluence/display/c/Parasoft) |  |  | Runtime analysis |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/c/Polyspace+Bug+Finder) | R2022b | [CERT C: Rule INT32-C](https://www.mathworks.com/help/bugfinder/ref/certcruleint32c.html) | Checks for:   * Integer overflow * Tainted division operand * Tainted modulo operand   Rule partially covered. |
| [PRQA QA-C](https://wiki.sei.cmu.edu/confluence/display/c/PRQA+QA-C) | 9.7 | **2800, 2801, 2802, 2803,**  **2860, 2861, 2862, 2863** | Fully implemented |
| [PRQA QA-C++](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046345) | 4.4 | **2800, 2801, 2802, 2803,**  **2860, 2861, 2862, 2863** |  |
| [PVS-Studio](https://wiki.sei.cmu.edu/confluence/display/cplusplus/PVS-Studio) | 7.23 | [V1026](https://pvs-studio.com/en/docs/warnings/v1026/)**,**[V1070](https://pvs-studio.com/en/docs/warnings/v1070/)**,**[V1081](https://pvs-studio.com/en/docs/warnings/v1081/)**,**[V1083](https://pvs-studio.com/en/docs/warnings/v1083/)**,**[V1085](https://pvs-studio.com/en/docs/warnings/v1085/)**,**[V5010](https://pvs-studio.com/en/docs/warnings/v5010/) |  |
| [TrustInSoft Analyzer](https://wiki.sei.cmu.edu/confluence/display/c/TrustInSoft+Analyzer) | 1.38 | **signed\_overflow** | Exhaustively verified (see [one compliant and one non-compliant example](https://taas.trust-in-soft.com/tsnippet/t/06486475)). |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Data Value Coding Standard** |
| --- | --- | --- |
| **Data Value** | [STD-MSC52-CPP] | Value returning functions must return a value from all exit paths |

| **Noncompliant Code** |
| --- |
| The code does not include the input value for positive input, so not all code paths will return a valid value. |
| int absolute\_value(int a) {  if (a < 0) {  return -a;  }  } |

| **Compliant Code** |
| --- |
| In this solution, there is a return value for all code paths, including positive input |
| int absolute\_value(int a) {  if (a < 0) {  return -a;  }  return a;  } |

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

| **Principles(s):**  **Architect and Design for Security Policies:** the software should be designed for security by ensuring that all possible exit paths will return the expected value.  **Adopt a Secure Coding Standard:** Make it a coding standard to always return a value from all exit paths to prevent any unexpected behavior. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Medium | Probable | Medium | P8 | L2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | **return-implicit** | Fully checked |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC++-MSC52** |  |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Clang) | 3.9 | **-Wreturn-type** | Does not catch all instances of this rule, such as function-try-blocks |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 7.2p0 | **LANG.STRUCT.MRS** | Missing return statement |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2022.4 | **DF2888** |  |
| [Klocwork](https://www.securecoding.cert.org/confluence/display/cplusplus/Klocwork) | 2022.4 | **FUNCRET.GEN**  **FUNCRET.IMPLICIT** |  |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | **2 D, 36 S** | Fully implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2022.2 | **CERT\_CPP-MSC52-a** | All exit paths from a function, except main(), with non-void return type shall have an explicit return statement with an expression |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2022b | [CERT C++: MSC52-CPP](https://www.mathworks.com/help/bugfinder/ref/certcmsc52cpp.html) | Checks for missing return statements (rule partially covered) |
| [SonarQube C/C++ Plugin](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046388) | 4.10 | [S935](https://www.sonarsource.com/products/codeanalyzers/sonarcfamilyforcpp/rules-cpp.html#RSPEC-935) |  |
| [PRQA QA-C++](https://www.securecoding.cert.org/confluence/pages/viewpage.action?pageId=142409849) | 4.4 | **1510** |  |
| [PVS-Studio](https://wiki.sei.cmu.edu/confluence/display/cplusplus/PVS-Studio) | 7.23 | [V591](https://pvs-studio.com/en/docs/warnings/v591/) |  |
| [RuleChecker](https://wiki.sei.cmu.edu/confluence/display/cplusplus/RuleChecker) | 22.10 | **return-implicit** | Fully checked |

#### Coding Standard 3

| **Coding Standard** | **Label** | **String Correctness Coding Standard** |
| --- | --- | --- |
| **String Correctness** | [STD-STR50-CPP] | Guarantee that storage for strings has sufficient space for character data and the null terminator. |

| **Noncompliant Code** |
| --- |
| The input is unbounded, so the code can produce a buffer overflow. |
| #include <iostream>    void f() {  char buf[12];  std::cin >> buf;  } |

| **Compliant Code** |
| --- |
| The solution is to use std::string instead of a bounded array to prevent a buffer overflow. |
| #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:** Ensure that strings are input properly and can be held in the proper storage.  **Heed Compiler Warnings:** The compiler may give a warning if a string is not properly implemented in storage. Do not ignore these, as it can cause a vulnerability.  **Use Effective Quality Assurance Techniques:** Use testing techniques to catch any possible overflow vulnerabilities. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | Medium | P18 | L1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | **stream-input-char-array** | Partially checked + soundly supported |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 7.2p0 | **MISC.MEM.NTERM**  **LANG.MEM.BO** **LANG.MEM.TO** | No space for null terminator  Buffer overrun Type overrun |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2022.4 | **C++5216**  **DF2835, DF2836, DF2839,** |  |
| [Klocwork](https://www.securecoding.cert.org/confluence/display/cplusplus/Klocwork) | 2022.4 | **NNTS.MIGHT** **NNTS.TAINTED** **NNTS.MUST** **SV.UNBOUND\_STRING\_INPUT.CIN** |  |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | **489 S, 66 X, 70 X, 71 X** | Partially implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2022.2 | **CERT\_CPP-STR50-b** **CERT\_CPP-STR50-c** **CERT\_CPP-STR50-e** **CERT\_CPP-STR50-f** **CERT\_CPP-STR50-g** | Avoid overflow due to reading a not zero terminated string Avoid overflow when writing to a buffer Prevent buffer overflows from tainted data Avoid buffer write overflow from tainted data Do not use the 'char' buffer to store input from 'std::cin' |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2022b | [CERT C++: STR50-CPP](https://www.mathworks.com/help/bugfinder/ref/certcstr50cpp.html) | Checks for:   * Use of dangerous standard function * Missing null in string array * Buffer overflow from incorrect string format specifier * Destination buffer overflow in string manipulation * Insufficient destination buffer size   Rule partially covered. |
| [RuleChecker](https://wiki.sei.cmu.edu/confluence/display/cplusplus/RuleChecker) | 22.10 | **stream-input-char-array** | Partially checked |
| [SonarQube C/C++ Plugin](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046388) | 4.10 | [S3519](https://www.sonarsource.com/products/codeanalyzers/sonarcfamilyforcpp/rules-cpp.html#RSPEC-3519) |  |

#### Coding Standard 4

| **Coding Standard** | **Label** | **SQL Injection Coding Standard** |
| --- | --- | --- |
| **SQL Injection** | [STD-STR02-C] | Sanitize data passed to complex subsystems. String data passed to complex subsystems may contain special characters that can trigger commands or actions |

| **Noncompliant Code** |
| --- |
| This code inputs an email address to a buffer and then uses the string as an argument to call to system(). This risks an SQL injection. |
| sprintf(buffer, "/bin/mail %s < /tmp/email", addr);  system(buffer); |

| **Compliant Code** |
| --- |
| All data that is entered is sanitized and validated before accepted and passed to the string |
| static char ok\_chars[] = "abcdefghijklmnopqrstuvwxyz"                           "ABCDEFGHIJKLMNOPQRSTUVWXYZ"                           "1234567890\_-.@";  char user\_data[] = "Bad char 1:} Bad char 2:{";  char \*cp = user\_data; /\* Cursor into string \*/  const char \*end = user\_data + strlen( user\_data);  for (cp += strspn(cp, ok\_chars); cp != end; cp += strspn(cp, ok\_chars)) {    \*cp = '\_';  } |

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

| **Principles(s):**  **Default Deny:** Make sure to implement permissions for users and deny a user access unless they meet the required conditions.  **Sanitize Data Sent to Other Systems:** Data needs to be sanitized to ensure that the subsystem will understand the context of the call and prevent an unwanted outcome. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | Medium | P18 | L1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152428) | 22.04 |  | Supported by stubbing/taint analysis |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 7.2p0 | **IO.INJ.COMMAND** **IO.INJ.FMT** **IO.INJ.LDAP** **IO.INJ.LIB** **IO.INJ.SQL** **IO.UT.LIB** **IO.UT.PROC** | Command injection Format string injection LDAP injection Library injection SQL injection Untrusted Library Load Untrusted Process Creation |
| [Coverity](https://wiki.sei.cmu.edu/confluence/display/c/Coverity) | 6.5 | **TAINTED\_STRING** | Fully implemented |
| [Klocwork](https://wiki.sei.cmu.edu/confluence/display/c/Klocwork) | 2022.4 | **NNTS.TAINTED** **SV.TAINTED.INJECTION** |  |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/c/LDRA) | 9.7.1 | **108 D, 109 D** | Partially implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/c/Parasoft) | 2022.2 | **CERT\_C-STR02-a** **CERT\_C-STR02-b** **CERT\_C-STR02-c** | Protect against command injection Protect against file name injection Protect against SQL injection |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/c/Polyspace+Bug+Finder) | R2022b | [CERT C: Rec. STR02-C](https://www.mathworks.com/help/bugfinder/ref/certcrec.str02c.html) | Checks for:   * Execution of externally controlled command * Command executed from externally controlled path * Library loaded from externally controlled path   Rec. partially covered. |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Memory Protection Coding Standard** |
| --- | --- | --- |
| **Memory Protection** | [STD-MEM52-CPP] | Detect and handle memory allocation errors. |

| **Noncompliant Code** |
| --- |
| An array of int is created and the results of the allocation are not checked. The function is marked as noexcept, so the caller assumes that the function will not throw any exceptions. This can lead to an unexpected termination of the program. |
| #include <cstring>    void f(const int \*array, std::size\_t size) noexcept {    int \*copy = new int[size];    std::memcpy(copy, array, size \* sizeof(\*copy));    // ...    delete [] copy;  } |

| **Compliant Code** |
| --- |
| Using std::nowthrow returns a null pointer or a pointer to the allocated space. The error is handled appropriately, and the system will not abnormally terminate. |
| #include <cstring>  #include <new>    void f(const int \*array, std::size\_t size) noexcept {  int \*copy = new (std::nothrow) int[size];  if (!copy) {  // Handle error  return;  }  std::memcpy(copy, array, size \* sizeof(\*copy));  // ...  delete [] copy;  } |

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

| **Principles(s):**  **Validate Input Data:** Make sure to validate any input which is allocated to memory.  **Architect and Design for Security Policies:** Always handle any possible errors using exceptions or catch/throws. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | Medium | P18 | L1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Compass/ROSE](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Rose) |  |  |  |
| [Coverity](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Coverity) | 7.5 | **CHECKED\_RETURN** | Finds inconsistencies in how function call return values are handled |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2022.4 | **C++3225, C++3226, C++3227, C++3228, C++3229, C++4632** |  |
| [Klocwork](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Klocwork) | 2022.4 | **NPD.CHECK.CALL.MIGHT** **NPD.CHECK.CALL.MUST** **NPD.CHECK.MIGHT** **NPD.CHECK.MUST** **NPD.CONST.CALL** **NPD.CONST.DEREF** **NPD.FUNC.CALL.MIGHT** **NPD.FUNC.CALL.MUST** **NPD.FUNC.MIGHT** **NPD.FUNC.MUST** **NPD.GEN.CALL.MIGHT** **NPD.GEN.CALL.MUST** **NPD.GEN.MIGHT** **NPD.GEN.MUST** **RNPD.CALL** **RNPD.DEREF** |  |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | **45 D** | Partially implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2022.2 | **CERT\_CPP-MEM52-a** **CERT\_CPP-MEM52-b** | Check the return value of new Do not allocate resources in function argument list because the order of evaluation of a function's parameters is undefined |
| [Parasoft Insure++](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) |  |  | Runtime detection |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2022b | [CERT C++: MEM52-CPP](https://www.mathworks.com/help/bugfinder/ref/certcmem52cpp.html) | Checks for unprotected dynamic memory allocation (rule partially covered) |
| [PRQA QA-C++](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046345) | 4.4 | **3225, 3226, 3227, 3228, 3229, 4632** |  |
| [PVS-Studio](https://wiki.sei.cmu.edu/confluence/display/cplusplus/PVS-Studio) | 7.23 | [V522](https://pvs-studio.com/en/docs/warnings/v522/)**,**[V668](https://pvs-studio.com/en/docs/warnings/v668/) |  |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Assertions Coding Standard** |
| --- | --- | --- |
| **Assertions** | [STD-DCL55-CPP] | Avoid information leakage when passing a class object across a trust boundary. |

| **Noncompliant Code** |
| --- |
| This code runs in kernel space and copies data from arg to the user space. Padding bits may be used within the object. These padding bits may contain sensitive information which could be leaked when data is copied to user space. |
| #include <cstddef>    struct test {  int a;  char b;  int c;  };    // Safely copy bytes to user space  extern int copy\_to\_user(void \*dest, void \*src, std::size\_t size);    void do\_stuff(void \*usr\_buf) {  test arg{1, 2, 3};  copy\_to\_user(usr\_buf, &arg, sizeof(arg));  } |

| **Compliant Code** |
| --- |
| By padding bits and adding the static\_assert() declaration, the expression is evaluated at compile time and, if false, the compilation is terminated, and the error message is used. |
| #include <cstddef>    struct test {  int a;  char b;  char padding\_1, padding\_2, padding\_3;  int c;    test(int a, char b, int c) : a(a), b(b),  padding\_1(0), padding\_2(0), padding\_3(0),  c(c) {}  };  // Ensure c is the next byte after the last padding byte.  static\_assert(offsetof(test, c) == offsetof(test, padding\_3) + 1,  "Object contains intermediate padding");  // Ensure there is no trailing padding.  static\_assert(sizeof(test) == offsetof(test, c) + sizeof(int),  "Object contains trailing padding");    // Safely copy bytes to user space.  extern int copy\_to\_user(void \*dest, void \*src, std::size\_t size);    void do\_stuff(void \*usr\_buf) {  test arg{1, 2, 3};  copy\_to\_user(usr\_buf, &arg, sizeof(arg));  } |

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

| **Principles(s):**  **Sanitize Data Sent to Other Systems:** When passing a class object, the data should be sanitized by padding bits and using static assertion. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Unlikely | High | P1 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC++-DCL55** |  |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 7.2p0 | **MISC.PADDING.POTB** | Padding Passed Across a Trust Boundary |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2022.4 | **DF4941, DF4942, DF4943** |  |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2022.2 | **CERT\_CPP-DCL55-a** | A pointer to a structure should not be passed to a function that can copy data to the user space |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/c/Polyspace+Bug+Finder) | R2022b | [CERT C++: DCL55-CPP](https://www.mathworks.com/help/bugfinder/ref/certcdcl55cpp.html) | Checks for information leakage due to structure padding (rule partially covered) |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Exceptions Coding Standard** |
| --- | --- | --- |
| **Exceptions** | [STD-ERR51-CPP] | Handle all exceptions. When an exception is thrown, control is transferred to the nearest handler with a type that matches the type of the exception thrown. If no matching handler is directly found, the search for a matching handler continues. |

| **Noncompliant Code** |
| --- |
| Neither of the functions catch exceptions thrown by throwing\_func() because no matching handler can be found for the exception thrown. This results in termination of the program. |
| void throwing\_func() noexcept(false);    void f() {  throwing\_func();  }    int main() {  f();  } |

| **Compliant Code** |
| --- |
| main() now handles all exceptions, which ensures that all exceptions thrown are handled and the program is not terminated. |
| void throwing\_func() noexcept(false);    void f() {  throwing\_func();  }    int main() {  try {  f();  } catch (...) {  // Handle error  }  } |

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

| **Principles(s):**  **Architect and Design for Security Policies:** Handle all possible errors by using exceptions and catch/throws. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Probable | Medium | P4 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | **main-function-catch-all** **early-catch-all** | Partially checked |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC++-ERR51** |  |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 7.2p0 | **LANG.STRUCT.UCTCH** | Unreachable Catch |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2022.4 | **C++4035, C++4036, C++4037** |  |
| [Klocwork](https://www.securecoding.cert.org/confluence/display/cplusplus/Klocwork) | 2022.4 | **MISRA.CATCH.ALL** |  |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | **527 S** | Partially implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2022.2 | **CERT\_CPP-ERR51-a** **CERT\_CPP-ERR51-b** | Always catch exceptions Each exception explicitly thrown in the code shall have a handler of a compatible type in all call paths that could lead to that point |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2022b | [CERT C++: ERR51-CPP](https://www.mathworks.com/help/bugfinder/ref/certcerr51cpp.html) | Checks for unhandled exceptions (rule partially covered) |
| [PRQA QA-C++](https://www.securecoding.cert.org/confluence/pages/viewpage.action?pageId=142409849) | 4.4 | **4035, 4036, 4037** |  |
| [RuleChecker](https://wiki.sei.cmu.edu/confluence/display/cplusplus/RuleChecker) | 22.10 | **main-function-catch-all** **early-catch-all** | Partially checked |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Input/Output Coding Standard** |
| --- | --- | --- |
| Input/Output | [STD-FIO50-CPP] | Do not alternatively input and output from a file stream without an intervening positioning call. |

| **Noncompliant Code** |
| --- |
| Data appends to the end of a file and then reads from the same file. Since there is no intervening positioning call between the output and input calls, the behavior in undefined. |
| #include <fstream>  #include <string>    void f(const std::string &fileName) {  std::fstream file(fileName);  if (!file.is\_open()) {  // Handle error  return;  }    file << "Output some data";  std::string str;  file >> str;  } |

| **Compliant Code** |
| --- |
| There is a function, seekg(), that is called between the output and input which eliminates the undefined behavior. |
| #include <fstream>  #include <string>    void f(const std::string &fileName) {  std::fstream file(fileName);  if (!file.is\_open()) {  // Handle error  return;  }    file << "Output some data";    std::string str;  file.seekg(0, std::ios::beg);  file >> str;  } |

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

| **Principles(s):**  **Validate Input Data:** Make sure to validate the input file when using I/O for a file stream to prevent any undefined behavior. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Unlikely | Medium | P6 | L2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion Bauhaus Suite | 7.2.0 | CERT\_CPP-FIO50 |  |
| CodeSonar | 7.2p0 | IO.IOWOP  IO.OIWOP | Input after output without positioning.  Output after input without positioning. |
| Helix QAC | 2022.4 | DF4711  DF4712  DF4713 |  |
| Parasoft C/C++test | 2022.2 | CERT\_CPP-FIO50-a | Do not alternately input and output from a stream without an intervening flush or positioning call |
| Polyspace Bug Finder | R2022b | CERT C++: FIO50-CPP | Checks for alternating input and output from a stream without flush or positioning call (rule fully covered) |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Declaration Coding Standard** |
| --- | --- | --- |
| Declaration | [STD-DLC52-CPP] | Never qualify a reference type with const or volatile. |

| **Noncompliant Code** |
| --- |
| The code declares p to be a reference to a const char. The following modification of p is not possible. |
| #include <iostream>    void f(char c) {  const char &p = c;  p = 'p'; // Error: read-only variable is not assignable  std::cout << c << std::endl;  } |

| **Compliant Code** |
| --- |
| The const qualifier is removed, so p can be modified. |
| #include <iostream>    void f(char c) {  char &p = c;  p = 'p';  std::cout << c << std::endl;  } |

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

| **Principles(s):**  **Keep It Simple:** Code should not be overcomplicated when qualifying a reference type so that it is easily understood and reduces the risk of a missed vulnerability.  **Validate Input Data:** Make sure to validate the reference type by correctly using qualifiers. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Unlikely | Low | P3 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion Bauhaus Suite | 7.2.0 | CertC++-DCL52 |  |
| Helix QAC | 2022.4 | C++0014 |  |
| Klocwork | 2022.4 | CERT.DCL.REF\_TYPE.CONST\_OR\_VOLATILE |  |
| Parasoft C/C++test | 2022.2 | CERT\_CPP-DCL52-a | Never qualify a reference type with ‘const’ or ‘volatile’ |
| Polyspace Bug Finder | R2022b | CERT C++: DCL52-CPP | Checks for: const-qualified reference types, Modification of const-qualified reference types. |
| PRQA QA-C++ | 4.4 | 0014 |  |
| Clang | 3.9 |  | Clang checks for violations of this rule and produces an error without the need to specify any special flags or options |
| SonarQube C/C++ Plugin | 4.10 | S3708 |  |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Containers Coding Standard** |
| --- | --- | --- |
| Containers | [STD-CTR51-CPP] | Use valid references, pointers, and iterators to reference elements of a container. |

| **Noncompliant Code** |
| --- |
| After the first call to insert(), pos is invalidated and the subsequent loop iterations will have undefined behavior. |
| #include <deque>    void f(const double \*items, std::size\_t count) {  std::deque<double> d;  auto pos = d.begin();  for (std::size\_t i = 0; i < count; ++i, ++pos) {  d.insert(pos, items[i] + 41.0);  }  } |

| **Compliant Code** |
| --- |
| The handwritten loop is replaced with the generic standard template library algorithm std::transform(). This accepts the range of elements to transform. |
| #include <algorithm>  #include <deque>  #include <iterator>    void f(const double \*items, std::size\_t count) {  std::deque<double> d;  std::transform(items, items + count, std::inserter(d, d.begin()),  [](double d) { return d + 41.0; });  } |

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

| **Principles(s):**  **Validate Input Data:** Make sure to validate any references, pointers, and iterators to prevent undefined behavior.  **Adopt a Secure Coding Standard:** Use a coding standard to ensure that you are always using valid references, pointers, and iterators when referencing elements of a container. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Probable | High | P6 | L2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.10 | Overflow\_upon\_dereference |  |
| CodeSonar | 7.2P0 | ALLOC.UAF | Use After Free |
| Helix QAC | 2022.4 | DF4746  DF4747  DF4748  DF4749 |  |
| Klocwork | 2022.4 | ITER.CONTAINER.MODIFIED |  |
| Parasoft C/C++test | 2022.2 | CERT\_CPP-CTR51-a | Do not modify a container while iterating over it |
| Polyspace Bug Finder | R2022b | CERT C++: CTR51-CPP | Checks for invalid use of iterator |
| PVS-Studio | 7.23 | V783 |  |

### 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.

From the very beginning, developers should be testing their code to ensure that no error can get through to the end and result in a vulnerability. By testing early and often, it is more likely that code will be secure and have no risk for vulnerabilities. While it is important to assess and plan in the pre-production phase, it is extremely important to verify and test code even during the early stages of development. Once in the production phase, transition and health checks should be performed and monitoring and detecting flaws should be continuously performed and responded to. At the end of production, the developers should make sure again to assess for any security flaws and maintain a secure program. Therefore, automation should take place throughout the entire development process.

Automation can be performed by using automation tools to enforce coding standards. These automation tools will provide the developer with testing methods to check their code for any possible vulnerabilities and tell the dev how to fix them. The developer could also use LDAP, or lightweight directory access protocol, to query user information rapidly. LDAP controls the access and maintenance of static data, such as usernames, passwords, email addresses, and other sensitive information. By implementing LDAP into the program, this does allow for a vulnerability unless SSL/TLS encryption is used as well. The developer could also use AD, or active directory, to store sensitive information. AD is a secure way to control access to network resources.

### 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-INT32-C | High | Likely | High | | 9 | 2 |
| STD-MSC52-CPP | Medium | Probable | Medium | | 8 | 2 |
| STD-STR50-CPP | High | Likely | Medium | | 18 | 1 |
| STD-STR02-C | High | Likely | Medium | | 18 | 1 |
| STD-MEM52-CPP | High | Likely | Medium | | 18 | 1 |
| STD-DCL55-CPP | Low | Unlikely | High | | 1 | 3 |
| STD-ERR51-CPP | Low | Probable | Medium | | 4 | 3 |
| STD-FIO50-CPP | Low | Unlikely | Medium | | 6 | 2 |
| STD-DLC52-CPP | Low | Unlikely | Low | | 3 | 3 |
| STD-CTR51-CPP | High | Probable | High | | 6 | 2 |

### Create Policies for Encryption and Triple A

Include all three types of encryption (in flight, at rest, and in use) and each of the three elements of the Triple-A framework using the tables provided***.***

* 1. Explain each type of encryption, how it is used, and why and when the policy applies.
  2. Explain each type of Triple-A framework strategy, how it is used, and why and when the policy applies.

Write policies for each and explain what it is, how it should be applied in practice, and why it should be used.

| 1. **Encryption** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Encryption in rest | Encryption at rest is used to protect data that is stored on a physical storage device, such as an SSD. By encrypting data on the disk, an attacker will not be able to view the data unless authorized to do so. |
| Encryption at flight | Encryption at flight occurs when data is encrypted while it is in the process of being transmitted through a network. This is most important for companies that use the open internet to transmit data. This prevents an attacker from intercepting to data and being able to view it. |
| Encryption in use | Encryption in use ensures that all sensitive data is protected at all times, regardless of use or location of data. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication only allows verified users to access data. This is used for most databases so that users can only access the data if they have been granted permission. A user will have to have a granted login to access a database or they will have to create a new account if allowed. |
| Authorization | Users are given specific permissions depending on their access level. Depending on access level, some users may only be able to view data, while others may be able to modify data within a database. |
| Accounting | It is beneficial to monitor user actions within the database. Accounting allows for tracking of things such as files that are accessed by users and any changes they made in the database. |

**\***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

|  |  |  |
| --- | --- | --- |
| Standard | Principle | Justification |
| STD-INT32-C | #1 Validate Input Data  #2 Heed Compiler  Warnings  #9 Use Effective Quality Assurance Techniques. | Make sure that input values are validated to ensure no overflows occur.  Check the compiler warnings for any data type errors.  Use testing methods to check the code for any possible overflow vulnerabilities. |
| STD-MSC52-CPP | #3 Architect and Design for Security Policies  #10 Adopt a Secure Coding Standard | The software should be designed for security by ensuring that all possible exit paths will return the expected value.  Make it a coding standard to always return a value from all exit paths to prevent any unexpected behavior. |
| STD-STR50-CPP | #1 Validate Input Data  #2 Heed Compiler Warnings  #9 Use Effective Quality Assurance Techniques | Ensure that strings are input properly and can be held in the proper storage.  The compiler may give a warning if a string is not properly implemented in storage.  Use testing techniques to catch any possible overflow vulnerabilities. |
| STD-STR02-C | #5 Default Deny  #7 Sanitize Data Sent to Other Systems | Make sure to implement permissions for users and deny a user access unless they meet the required conditions.  Data needs to be sanitized to ensure that the subsystem will understand the context of the call and prevent an unwanted outcome. |
| STD-MEM52-CPP | #1 Validate Input Data  #3 Architect and Design for Security Policies | Make sure to validate any input which is allocated to memory.  Always handle any possible errors using exceptions or catch/throws. |
| STD-DCL55-CPP | #7 Sanitize Data Sent to Other Systems | When passing a class object, the data should be sanitized by padding bits and using static assertion. |
| STD-ERR51-CPP | #3 Architect and Design for Security Policies | Handle all possible errors by using exceptions and catch/throws. |
| STD-FIO50-CPP | #1 Validate Input Data | Make sure to validate the input file when using I/O for a file stream to prevent any undefined behavior. |
| STD-DLC52-CPP | #4 Keep It Simple  #1 Validate Input Data | Code should not be overcomplicated when qualifying a reference type so that it is easily understood and reduces the risk of a missed vulnerability.  Make sure to validate the reference type by correctly using qualifiers. |
| STD-CTR51-CPP | #1 Validate Input Data  #10 Adopt a Secure Coding Standard | Make sure to validate any references, pointers, and iterators to prevent undefined behavior.  Use a coding standard to ensure that you are always using valid references, pointers, and iterators when referencing elements of a container. |

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 | 02/11/2023 | First Revision | Jaelyn Sloan |  |
| [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 |