**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 6](#_Toc52464060)

[Coding Standard 3 8](#_Toc52464061)

[Coding Standard 4 10](#_Toc52464062)

[Coding Standard 5 12](#_Toc52464063)

[Coding Standard 6 14](#_Toc52464064)

[Coding Standard 7 16](#_Toc52464065)

[Coding Standard 8 18](#_Toc52464066)

[Coding Standard 9 20](#_Toc52464067)

[Coding Standard 10 22](#_Toc52464068)

[Defense-in-Depth Illustration 24](#_Toc52464069)

[Project One 24](#_Toc52464070)

[1. Revise the C/C++ Standards 24](#_Toc52464071)

[2. Risk Assessment 24](#_Toc52464072)

[3. Automated Detection 24](#_Toc52464073)

[4. Automation 24](#_Toc52464074)

[5. Summary of Risk Assessments 25](#_Toc52464075)

[6. Create Policies for Encryption and Triple A 26](#_Toc52464076)

[7. Map the Principles 27](#_Toc52464077)

[Audit Controls and Management 28](#_Toc52464078)

[Enforcement 28](#_Toc52464079)

[Exceptions Process 28](#_Toc52464080)

[Distribution 29](#_Toc52464081)

[Policy Change Control 29](#_Toc52464082)

[Policy Version History 29](#_Toc52464083)

[Appendix A Lookups 29](#_Toc52464084)

[Approved C/C++ Language Acronyms 29](#_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 | As the first line of defense against injection attacks, input validation ensures data is sanitized and validated against strict criteria. Validating input data prevents attackers from manipulating system behavior through malicious inputs. |
| 1. Heed Compiler Warnings | Compiler warnings often can reveal bugs that can escalate into vulnerabilities and should be treated as errors to catch potential vulnerabilities early on. |
| 1. Architect and Design for Security Policies | Integrate security into the software development life cycle. Embedding security from the design phase and incorporating them throughout the cycle can reduce costs and prevent architectural flaws. |
| 1. Keep It Simple | Reduce complexity to minimize attack surfaces, as complexity can breed vulnerabilities. |
| 1. Default Deny | Restrict access by default, only grant permissions explicitly as needed. |
| 1. Adhere to the Principle of Least Privilege | Limit user/system access to what is necessary. |
| 1. Sanitize Data Sent to Other Systems | Prevent data leaks via output encoding/validation. |
| 1. Practice Defense in Depth | Use Multiple security layers to protect against single-point failures so the entire system isn’t compromised. |
| 1. Use Effective Quality Assurance Techniques | Combine automated and manual testing by implementing SAST/DAST, fuzz testing, and peer reviews. |
| 1. Adopt a Secure Coding Standard | Follow CERT C++ or OWASP Guidelines. |

### 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.

Hickmon, J. (2024, December 11). *Multimodal Approaches to Fair Image Classification: An Ethical Perspective*. arXiv. https://doi.org/10.48550/arXiv.2412.12165

#### Coding Standard 1

| **Coding Standard** | **Label** | **Do not write syntactically ambiguous declarations** |
| --- | --- | --- |
| **Data Type** | [STD-001-CPP] | Having ambiguous declarations can lead to misunderstandings and misinterpretations for developers and compilers and can cause logical errors. This standard ensures clarity in variable and function declarations by avoiding constructs that could be parsed in multiple ways. |

| **Noncompliant Code** |
| --- |
| A simple piece of code below can be misinterpreted and cause errors because of the missing parentheses. In this code, int\* a[10] can be understood as an array of 10 integer pointers rather than a single pointer to an array of integers. This ambiguity can lead to incorrect assumptions about memory layout and cause unintended behavior. |
| int\* a[10]; |

| **Compliant Code** |
| --- |
| The code below uses the parentheses to clarify intent. It explicitly declares a pointer to an array of 10 integers, and eliminates any ambiguity. |
| Int (\*a) [10]; |

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

| **Principles(s):**  Keep It Simple – Avoid complex declarations that could be misinterpreted.  Adopt a Secure Coding Standard – Follow CERT C++ guidelines for clear syntax.  Heed Compiler Warnings – Enable warnings to catch ambiguous declarations early. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Clang-Tidy | 12.0 | readability-avoid-ambiguous-declarations | Detects syntactically ambiguous declarations and fixes typical programming errors. Clang-Tidy enforces CERT C++ Rule DCL53-CPP by flagging ambiguous pointer/array syntax |
| CppCheck | 2.7 | --warning=ambiguousDeclaration | Warns about potentially unclear declarations. |
| PVS-Studio | 7.23 | V706 | Flags ambiguous declarations, bugs, and potential vulnerabilities. This detects syntactically ambiguous declarations that could lead to logical errors. |

Coding Standard 2

| **Coding Standard** | **Label** | **Do not read uninitialized memory** |
| --- | --- | --- |
| **Data Value** | [STD-002-CPP] | Reading uninitialized memory leads to undefined behavior, which can cause crashes, incorrect program logic, or security vulnerabilities. This standard ensures variables are properly initialized before use. |

| **Noncompliant Code** |
| --- |
| This section of code declares an integer x, but does not initialize it before reading its value. The integer x therefore contains an indeterminate value, and can result in undefined behavior. |
| int x;  int y = x; |

| **Compliant Code** |
| --- |
| Below, the integer x is initialized before it is used which ensures predictable behavior. Explicitly initializing values prevents undefined behavior. |
| int x = 0;  int y = x; |

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

| **Principles(s):**  Validate Input Data – Ensure variables are initialized before use.  Defensive Programming – Assume memory is unsafe until explicitly initialized. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Probable | Medium | High | 1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | **uninitialized-read** | Partially checked |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Clang) | 3.9 | -Wuninitialized  clang-analyzer-core.UndefinedBinaryOperatorResult | Does not catch all instances of this rule, such as uninitialized values read from heap-allocated memory. |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 9.0p0 | **LANG.STRUCT.RPL LANG.MEM.UVAR** | Return pointer to local Uninitialized variable |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2024.4 | **DF726, DF2727, DF2728, DF2961, DF2962, DF2963, DF2966, DF2967, DF2968, DF2971, DF2972, DF2973, DF2976, DF2977, DF978** |  |
| [Klocwork](https://www.securecoding.cert.org/confluence/display/cplusplus/Klocwork) | 2024.4 | **UNINIT.CTOR.MIGHT** **UNINIT.CTOR.MUST** **UNINIT.HEAP.MIGHT** **UNINIT.HEAP.MUST** **UNINIT.STACK.ARRAY.MIGHT** **UNINIT.STACK.ARRAY.MUST** **UNINIT.STACK.ARRAY.PARTIAL.MUST** **UNINIT.STACK.MIGHT** **UNINIT.STACK.MUST** |  |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | **53 D, 69 D, 631 S, 652 S** | Partially implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2024.2 | **CERT\_CPP-EXP53-a** | Avoid use before initialization. This checks for variables used before initialization. |
| [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) | R2024b | [CERT C++: EXP53-CPP](https://www.mathworks.com/help/bugfinder/ref/certcexp53cpp.html) | Checks for:   * Non-initialized variable * Non-initialized pointer   Rule partially covered. |
| [PVS-Studio](https://wiki.sei.cmu.edu/confluence/display/cplusplus/PVS-Studio) | 7.36 | [**V546**](https://pvs-studio.com/en/docs/warnings/v546/), [**V573**](https://pvs-studio.com/en/docs/warnings/v573/), [**V614**](https://pvs-studio.com/en/docs/warnings/v614/), [**V670**](https://pvs-studio.com/en/docs/warnings/v670/), [**V679**](https://pvs-studio.com/en/docs/warnings/v679/), [**V730**](https://pvs-studio.com/en/docs/warnings/v730/), [**V788**](https://pvs-studio.com/en/docs/warnings/v788/), [**V1007**](https://pvs-studio.com/en/docs/warnings/v1007/), [**V1050**](https://pvs-studio.com/en/docs/warnings/v1050/) |  |
| [RuleChecker](https://wiki.sei.cmu.edu/confluence/display/cplusplus/RuleChecker) | 22.10 | **uninitialized-read** | Partially checked |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Range Check element access** |
| --- | --- | --- |
| **String Correctness** | [STD-003-CPP] | Accessing elements outside valid ranges, like arrays, strings, containers, can lead to undefined behavior, crashes, or security vulnerabilities, like buffer overflows. This standard enforces bounds checking to ensure safe memory access. |

| **Noncompliant Code** |
| --- |
| This code attempts to access an element beyond the vector's bounds using an unchecked iterator arithmetic operation (begin() + 5). If the vector has fewer than 5 elements, this leads to undefined behavior. |
| #include <vector>  Void processElement(const std::vector<int>& vec) {  int value = \*(vec.begin() +5);  } |

| **Compliant Code** |
| --- |
| The compliant solution explicitly checks the vector size before dereferencing the iterator. This ensures safe access and prevents undefined behavior. |
| #include <vector>  Void processElement(const std::vector<int>& vec) {  If (vec.size() > 5 {  int value = \* (vec.begin() + 5);  }  } |

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

| **Principles(s):**  Practice Defense in Depth – Use bounds checks as a safety layer.  Default Deny – Assume indices are invalid until proven safe.  Architect for Security Policies – Enforce bounds checking in design. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | **assert\_failure** |  |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 9.0p0 | **LANG.MEM.BO** **LANG.MEM.BU** **LANG.MEM.TBA** **LANG.MEM.TO** **LANG.MEM.TU** | Buffer overrun Buffer underrun Tainted buffer access Type overrun Type underrun  This detects bugger overruns due to missing bounds checks. |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2024.4 | **C++3162, C++3163, C++3164, C++3165** |  |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2024.2 | **CERT\_CPP-STR53-a** | Guarantee that container indices are within the valid range |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2024b | [CERT C++: STR53-CPP](https://www.mathworks.com/help/bugfinder/ref/certcstr53cpp.html) | Checks for:   * Array access out of bounds * Array access with tainted index * Pointer dereference with tainted offset   Rule partially covered.  This flags array/container access with tainted indices. |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Prevent SQL Injection** |
| --- | --- | --- |
| **SQL Injection** | [STD-004-CPP] | SQL injection attacks occur when untrusted input is concatenated directly into SQL queries, allowing attackers to execute arbitrary commands. Parameterized queries separate data from code, eliminating injection risks. |

| **Noncompliant Code** |
| --- |
| The noncompliant code dynamically constructs an SQL query by concatenating user input (userInput) directly into the string. This allows malicious input (e.g., "admin'; DROP TABLE users--") to alter the query logic. |
| std::string userInput = getUserInput();  std::string query = "SELECT \* FROM users WHERE username = '" + userInput + "'";  executeQuery(query); |

| **Compliant Code** |
| --- |
| The compliant code uses parameterized queries (via placeholders like ? or :param), ensuring user input is treated as data rather than executable SQL. This prevents injection regardless of input content. |
| std::string userInput = getUserInput();  std::string query = "SELECT \* FROM users WHERE username = ?";  executeParameterizedQuery(query, userInput); |

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

| **Principles(s):** [  Validate Input Data – Treat all inputs as untrusted. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likley | Medium | High | 1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [The Checker Framework](https://wiki.sei.cmu.edu/confluence/display/java/The+Checker+Framework) | 2.1.3 | **Tainting Checker** | Trust and security errors (see Chapter 8) |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/java/CodeSonar) | 9.0p0 | **JAVA.IO.INJ.SQL** | SQL injection |
| [Coverity](https://wiki.sei.cmu.edu/confluence/display/java/Coverity) | 7.5 | **SQLI FB.SQL\_PREPARED\_STATEMENT\_GENERATED\_** **FB.SQL\_NONCONSTANT\_STRING\_PASSED\_TO\_EXECUTE** | Implemented |
| [Findbugs](https://wiki.sei.cmu.edu/confluence/display/java/Findbugs) | 1.0 | **SQL\_NONCONSTANT\_STRING\_PASSED\_TO\_EXECUTE** | Implemented |
| [Fortify](https://wiki.sei.cmu.edu/confluence/display/java/Fortify) | 1.0 | **HTTP\_Response\_Splitting** **SQL\_Injection\_\_Persistence** **SQL\_Injection** | Implemented |
| [Klocwork](https://wiki.sei.cmu.edu/confluence/display/java/Klocwork) | 2024.4 | **SV.DATA.DB** **SV.SQL** **SV.SQL.DBSOURCE** | Implemented |
| [Parasoft Jtest](https://wiki.sei.cmu.edu/confluence/display/java/Parasoft) | 2024.2 | **CERT.IDS00.TDSQL** | Protect against SQL injection |
| [SonarQube](https://wiki.sei.cmu.edu/confluence/display/java/SonarQube) | 9.9 | [**S2077**](https://rules.sonarsource.com/java/RSPEC-2077)  [**S3649**](https://rules.sonarsource.com/java/RSPEC-3649) | [Executing SQL queries is security-sensitive](https://rules.sonarsource.com/java/RSPEC-2077)  [SQL queries should not be vulnerable to injection attacks](https://rules.sonarsource.com/java/RSPEC-3649)  Identifies concatenated SQL queries vulnerable to injection. |
| [SpotBugs](https://wiki.sei.cmu.edu/confluence/display/java/SpotBugs) | 4.6.0 | **SQL\_NONCONSTANT\_STRING\_PASSED\_TO\_EXECUTE** **SQL\_PREPARED\_STATEMENT\_GENERATED\_FROM\_NONCONSTANT\_STRING** | Implemented |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Detect and Handle Memory Allocation Errors** |
| --- | --- | --- |
| **Memory Protection** | [STD-005-CPP] | Memory allocation failures can lead to crashes, undefined behavior, or security vulnerabilities if not handled. This standard ensures programs gracefully handle out-of-memory conditions and fail safely. |

| **Noncompliant Code** |
| --- |
| The noncompliant code allocates memory with new but fails to check if the allocation succeeded. If new throws std::bad\_alloc (or returns nullptr for nothrow), the program may crash when dereferencing the pointer. |
| int\* ptr = new int[100];  ptr[0] = 42; |

| **Compliant Code** |
| --- |
| The compliant code uses a try-catch block to handle std::bad\_alloc or checks nothrow allocations. This ensures safe degradation when memory is exhausted. |
| try {  int\* ptr = new int[100];  ptr[0] = 42;  } catch (const std::bad\_alloc&) {  std::cerr << "Memory allocation failed\n";  } |

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

| **Principles(s):**  Keep It Simple – Use RAII.  Adopt a Secure Coding Standard – Follow CERT C++ Rule MEM52-CPP. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | Medium | High | 1 |

**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) | 2024.4 | **C++3225, C++3226, C++3227, C++3228, C++3229, C++4632** | Enforces checking new return values. |
| [Klocwork](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Klocwork) | 2024.4 | **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) | 2024.2 | **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) | R2024b | [CERT C++: MEM52-CPP](https://www.mathworks.com/help/bugfinder/ref/certcmem52cpp.html) | Checks for unprotected dynamic memory allocation (rule partially covered) |
| [PVS-Studio](https://wiki.sei.cmu.edu/confluence/display/cplusplus/PVS-Studio) | 7.36 | [**V522**](https://pvs-studio.com/en/docs/warnings/v522/)**,** [**V668**](https://pvs-studio.com/en/docs/warnings/v668/) | Flags unchecked bad\_alloc exceptions. |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Use a Static assertion to test the value of a constant expression** |
| --- | --- | --- |
| **Assertions** | [STD-006-CPP] | Static assertions enforce requirements at compile-time rather than runtime, catching errors earlier in development. This prevents invalid configurations from becoming runtime failures. |

| **Noncompliant Code** |
| --- |
| The noncompliant code checks a constant expression condition at runtime using a standard assert, which wastes resources and delays error detection until execution. Runtime checks for constant conditions are unnecessary and inefficient. |
| #define MAX\_SIZE 10  assert(MAX\_SIZE > 5); |

| **Compliant Code** |
| --- |
| The compliant code uses static\_assert to validate the constant expression at compile-time. This ensures immediate feedback during compilation and eliminates runtime overhead for invariant checks. |
| #define MAX\_SIZE 10  static\_assert(MAX\_SIZE > 5, "MAX\_SIZE must be greater than 5"); |

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

| **Principles(s):**  Heed Compiler Warnings – Catch errors at compile-time.  Keep It Simple – Replace runtime checks with compile-time checks.  Adopt a Secure Coding Standard – Follow CERT C++ Rule DCL03-CPP. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/c/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC-DCL03** |  |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/c/Clang) | 3.9 | misc-static-assert | Checked by clang-tidy. Recommends static\_assert over assert for constrains. |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 9.0p0 | **(customization)** | Users can implement a custom check that reports uses of the assert() macro |
| [Compass/ROSE](https://wiki.sei.cmu.edu/confluence/display/c/Rose) |  |  | Could detect violations of this rule merely by looking for calls to assert(), and if it can evaluate the assertion (due to all values being known at compile time), then the code should use static-assert instead; this assumes ROSE can recognize macro invocation |
| [ECLAIR](https://wiki.sei.cmu.edu/confluence/display/c/ECLAIR) | 1.2 | **CC2.DCL03** | Fully implemented |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/c/LDRA) | 9.7.1 | **44 S** | Fully implemented |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Handle all exceptions** |
| --- | --- | --- |
| **Exceptions** | [STD-007-CPP] | Unhandled exceptions can cause program termination, data corruption, or undefined behavior. This standard ensures graceful recovery or controlled shutdown when exceptions occur. |

| **Noncompliant Code** |
| --- |
| The noncompliant code calls a function that may throw exceptions but provides no exception handling. If an exception propagates to main(), the program will terminate abruptly without cleanup. |
| void riskyOperation() { /\* May throw std::runtime\_error \*/ }  int main() {  riskyOperation();  return 0;  } |

| **Compliant Code** |
| --- |
| The compliant code wraps the risky operation in a try-catch block, handling exceptions explicitly. This ensures resources are cleaned up and users receive meaningful error messages. |
| void riskyOperation() { /\* May throw std::runtime\_error \*/ }  int main() {  try {  riskyOperation();  } catch (const std::exception& e) {  std::cerr << "Error: " << e.what() << std::endl;  return 1;  }  return 0;  } |

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

| **Principles(s):**  Architect for Security Policies – Plan exception-handling strategies. |
| --- |

**Threat Level**

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

**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/cplusplus/CodeSonar) | 9.0p0 | **LANG.STRUCT.UCTCH** | Unreachable Catch |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2024.4 | **C++4035, C++4036, C++4037** |  |
| [Klocwork](https://www.securecoding.cert.org/confluence/display/cplusplus/Klocwork) | 2024.4 | **MISRA.CATCH.ALL** | Ensures exceptions are caught in main(). |
| [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) | 2024.2 | **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) | R2024b | [CERT C++: ERR51-CPP](https://www.mathworks.com/help/bugfinder/ref/certcerr51cpp.html) | Checks for unhandled exceptions (rule partially covered) |
| [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** | **Do not modify the standard namespaces** |
| --- | --- | --- |
| Declarations and Initialization | [STD-008-CPP] | Adding to or modifying the std namespace (or other standard namespaces) leads to undefined behavior per the C++ standard. This prevents conflicts with current/future library implementations and ensures portability. |

| **Noncompliant Code** |
| --- |
| The noncompliant code attempts to add a custom specialization to the std namespace. This violates the C++ standard and may cause conflicts with compiler implementations or future library versions. |
| #include <utility>  namespace std {  template<>  void swap(MyType& a, MyType& b) {  a.customSwap(b);  }  } |

| **Compliant Code** |
| --- |
| The compliant code defines the custom swap in the same namespace as MyType instead. This allows argument-dependent lookup (ADL) to work correctly without violating namespace rules. |
| class MyType {  public:  void customSwap(MyType& other) { /\* ... \*/ }  };  void swap(MyType& a, MyType& b) {  a.customSwap(b);  } |

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

| **Principles(s):**  Keep It Simple – Avoid non-standard extensions.  Adopt a Secure Coding Standard – Follow CERT C++ Rule DCL58-CPP.  Heed Compiler Warnings – Enable -Wpedantic for namespace violations. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC++-DCL58** |  |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 9.0p0 | **LANG.STRUCT.DECL.SNM** | Modification of Standard Namespaces |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2024.4 | **C++3180, C++3181, C++3182** |  |
| [Klocwork](https://www.securecoding.cert.org/confluence/display/cplusplus/Klocwork) | 2024.4 | **CERT.DCL.STD\_NS\_MODIFIED** |  |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2024.2 | **CERT\_CPP-DCL58-a** | Do not modify the standard namespaces 'std' and 'posix' |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2024b | [CERT C++: DCL58-CPP](https://www.mathworks.com/help/bugfinder/ref/certcdcl58cpp.html) | Checks for modification of standard namespaces (rule fully covered) |
| [PVS-Studio](https://wiki.sei.cmu.edu/confluence/display/cplusplus/PVS-Studio) | 7.36 | [**V1061**](https://pvs-studio.com/en/docs/warnings/v1061/) |  |
| [SonarQube C/C++ Plugin](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046388) | 4.10 | [**S3470**](https://www.sonarsource.com/products/codeanalyzers/sonarcfamilyforcpp/rules-cpp.html#RSPEC-3470) | Flags modifications to std namespace. |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Write the constructor member initializers in the canonical order** |
| --- | --- | --- |
| Object Oriented Programming | [STD-009-CPP] | Members are initialized in the order they're declared in the class. Following this canonical order prevents subtle bugs where initialization depends on uninitialized members. |

| **Noncompliant Code** |
| --- |
| The noncompliant code initializes members in an arbitrary order in the constructor's initializer list. This can lead to undefined behavior if m\_value depends on m\_base being initialized first, since m\_base is declared after m\_value in the class. |
| class Example {  int m\_value;  int m\_base;  public:  Example(int v)  : m\_base(10),  m\_value(v + m\_base) {}  }; |

| **Compliant Code** |
| --- |
| The compliant code matches the initializer list order to the members' declaration order. This ensures safe initialization and makes the code's behavior explicit and predictable. |
| class Example {  int m\_value;  int m\_base;  public:  Example(int v)  : m\_value(v + 10),  m\_base(10) {}  }; |

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

| **Principles(s):**  Keep It Simple – Match initialization to declaration order.  Defensive Programming – Prevent dependency bugs.  Adopt a Secure Coding Standard – Follow CERT C++ Rule OOP53-CPP. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | **initializer-list-order** | Fully checked |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC++-OOP53** |  |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Clang) | 3.9 | -Wreorder | Warns about out-of-order initializers |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 9.0p0 | **LANG.STRUCT.INIT.OOMI** | Out of Order Member Initializers |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2024.4 | **C++4053** |  |
| [Klocwork](https://www.securecoding.cert.org/confluence/display/cplusplus/Klocwork) | 2024.4 | **CERT.OOP.CTOR.INIT\_ORDER** |  |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | **206 S** | Fully implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2024.2 | **CERT\_CPP-OOP53-a** | List members in an initialization list in the order in which they are declared |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2024b | [CERT C++: OOP53-CPP](https://www.mathworks.com/help/bugfinder/ref/certcoop53cpp.html) | Checks for members not initialized in canonical order (rule fully covered) |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Close Files when they are no longer needed** |
| --- | --- | --- |
| Input/ Output | [STD-010-CPP] | Unclosed files can lead to resource leaks, data corruption, or locked files that prevent other processes from accessing them. This standard ensures proper cleanup of file resources. |

| **Noncompliant Code** |
| --- |
| The noncompliant code opens a file but fails to explicitly close it. If an exception occurs before reaching the implicit close, the file handle leaks, potentially causing resource exhaustion or data integrity issues. |
| #include <fstream>  void processFile() {  std::fstream file("data.txt");  // ... use file ...  // File closed only when 'file' goes out of scope  } |

| **Compliant Code** |
| --- |
| The compliant code uses RAII (by limiting file scope) or explicitly closes the file when done. This ensures timely release of resources even if exceptions occur. |
| #include <fstream>  void processFile() {  { // Block scope for RAII  std::fstream file("data.txt");  // ... use file ...  file.close();  } |

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

| **Principles(s):**  Resource Acquisition Is Initialization (RAII) – Tie resource lifetime to scope.  Default Deny – Release resources unless explicitly retained.  Adopt a Secure Coding Standard – Follow CERT C++ Rule FIO51-CPP. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 9.0p0 | **ALLOC.LEAK** | Leak |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2024.4 | **DF4786, DF4787, DF4788** |  |
| [Klocwork](https://www.securecoding.cert.org/confluence/display/cplusplus/Klocwork) | 2024.4 | **RH.LEAK** |  |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2024.2 | **CERT\_CPP-FIO51-a** | Ensure resources are freed |
| [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) | R2024b | [CERT C++: FIO51-CPP](https://www.mathworks.com/help/bugfinder/ref/certcfio51cpp.html) | Checks for resource leak (rule partially covered). Detects file handle leaks. |

### Defense-in-Depth Illustration

This illustration provides a visual representation of the defense-in-depth best practice of layered security.



### Automation



Automation will be used for the enforcement of and compliance to the standards defined in this policy. Green Pace already has a well-established DevOps process and infrastructure. Define guidance on where and how to modify the existing DevOps process to automate enforcement of the standards in this policy. Use the DevSecOps diagram and provide an explanation using that diagram as context.

[Insert your written explanations here.]

### Summary of Risk Assessments

Consolidate all risk assessments into one table including both coding and systems standards, ordered by standard number.

| Rule | Severity | Likelihood | Remediation Cost | Priority | Level |
| --- | --- | --- | --- | --- | --- |
| STD-001-CPP | Low | Unlikely | Medium | Low | 3 |
| STD-002-CPP | High | Probable | Medium | High | 1 |
| STD-003-CPP | High | Unlikely | Medium | Medium | 2 |
| STD-004-CPP | High | Likely | Medium | High | 1 |
| STD-005-CPP | High | Likely | Medium | High | 1 |
| STD-006-CPP | Low | Unlikely | High | Low | 3 |
| STD-007-CPP | Low | Probable | Medium | Medium | 3 |
| STD-008-CPP | High | Unlikely | Medium | Medium | 2 |
| STD-009-CPP | Medium | Unlikely | Medium | Medium | 3 |
| STD-010-CPP | Medium | Unlikely | Medium | Medium | 3 |

### 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 | Encryption in rest protects stored data, like databases, backups, and files. The data is encrypted when written to disk and decrypted upon authorized access. This prevents data breaches from physical theft or unauthorized access. The keep it simple and least privilege principles can be applied here as using standardized algorithms and decrypting only for authorized users helps protect data. |
| Encryption in flight | Encryption in flight protects data that is moving, this is seen when data is moving between different devices in a network or out of the network. It is applied to web applications, HTTPS, API communications, and remote desktop sessions. This is common when encrypting data between clients and servers. This mitigates man-in-the-middle attacks and is essential for all external communications and internal sensitive traffic. |
| Encryption in use | Encryption in use protects data that is being processed, created, or edited. This can be applied to cloud workloads, in RAM, financial transactions, and medical data processing. This prevents memory-scrapping attacks and is critical for high-risk environments like in healthcare. Principles that should be incorporated are architect for security and default deny to help design a secure processing and to ensure decryption is only done in trusted environments. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication verifies user identity through certifications, biometrics, and passwords to ensure they are who is trying to log in. This logs all login attempts and helps prevent unauthorized access. This is required for all systems handling sensitive data. |
| Authorization | Authorization grants permissions and privileges based on different roles or attributes. This is needed to access files, access specific user levels, and make database changes. |
| Accounting | Accounting logs, tracks, and audits actions. This can collect timestamps, track logins, firewall logs, OS logs, and accessing files within a system. It can also help in supporting forensic investigations and is required. |

**\***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 | 04/13/2025 | Milestone Three | Ericka Resendez |  |
| 1.2 | 04/25/2025 | Project One | Ericka Resendez |  |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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