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



# Green Pace Secure Development Policy

# Courtney Warner

## Contents

[Green Pace Secure Development Policy 0](#_Toc200915258)

[Courtney Warner 0](#_Toc200915259)

[Contents 1](#_Toc200915260)

[Overview 3](#_Toc200915261)

[Purpose 3](#_Toc200915262)

[Scope 3](#_Toc200915263)

[Module Three Milestone 3](#_Toc200915264)

[Ten Core Security Principles 3](#_Toc200915265)

[C/C++ Ten Coding Standards 4](#_Toc200915266)

[Coding Standard 1 5](#_Toc200915267)

[Coding Standard 2 6](#_Toc200915268)

[Coding Standard 3 7](#_Toc200915269)

[Coding Standard 4 9](#_Toc200915270)

[Coding Standard 5 11](#_Toc200915271)

[Coding Standard 6 13](#_Toc200915272)

[Coding Standard 7 14](#_Toc200915273)

[Coding Standard 8 16](#_Toc200915274)

[Coding Standard 9 18](#_Toc200915275)

[Coding Standard 10 20](#_Toc200915276)

[Defense-in-Depth Illustration 22](#_Toc200915277)

[Project One 22](#_Toc200915278)

[Revise the C/C++ Standards 22](#_Toc200915279)

[Risk Assessment 22](#_Toc200915280)

[Automated Detection 22](#_Toc200915281)

[Automation 22](#_Toc200915282)

[Summary of Risk Assessments 23](#_Toc200915283)

[Create Policies for Encryption and Triple A 24](#_Toc200915284)

[Map the Principles 24](#_Toc200915285)

[Audit Controls and Management 26](#_Toc200915286)

[Enforcement 26](#_Toc200915287)

[Exceptions Process 26](#_Toc200915288)

[Distribution 27](#_Toc200915289)

[Policy Change Control 27](#_Toc200915290)

[Policy Version History 27](#_Toc200915291)

[Appendix A Lookups 27](#_Toc200915292)

[Approved C/C++ Language Acronyms 27](#_Toc200915293)

## Overview

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

## Purpose

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

## Scope

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

## Module Three Milestone

### Ten Core Security Principles

| **Principles** | Write a short paragraph explaining each of the 10 principles of security. |
| --- | --- |
| 1. ValidateInput Data | All users and external input must be validated to ensure it meets expected formats and does not contain malicious content. This helps prevent vulnerabilities like SQL injection and buffer overflows. |
| 1. Heed Compiler Warnings | Compiler warnings often indicate unsafe or questionable coding practices. Addressing these warnings early can prevent security flaws and runtime errors from entering production code. |
| 1. Architect and Design for Security Policies | Security should be considered from the earliest stages of system design, not just during implementation. Incorporating security policies into architecture helps reduce overall attack surfaces. |
| 1. Keep It Simple | Simple code is easier to understand, maintain, and audit for security issues. Complexity increases the likelihood of introducing bugs and vulnerabilities. |
| 1. Default Deny | Systems should deny access by default and only allow it when explicitly permitted. This limits unauthorized access and protects sensitive resources. |
| 1. Adhere to the Principle of Least Privilege | Users and processes should operate with the minimum privileges necessary to perform their tasks. This reduces the damage that can occur if an account or component is compromised. |
| 1. Sanitize Data Sent to Other Systems | Before sending data to external systems like databases or browsers, it should be sanitized to remove or neutralize harmful input. This helps prevent injection attacks and cross-site scripting (XSS). |
| 1. Practice Defense in Depth | Multiple layers of security should be used to protect systems, so if one layer fails, others remain in place. This approach increases overall system resilience to attacks. |
| 1. Use Effective Quality Assurance Techniques | Security flaws can often be identified through code reviews, static analysis, and rigorous testing. Including QA throughout development helps catch issues early and improves code quality. |
| 1. Adopt a Secure Coding Standard | Using established secure coding standards like SEI CERT C++ helps maintain consistency and reduce vulnerabilities. These guidelines offer proven practices for writing safe, reliable code. |

### C/C++ Ten Coding Standards

Complete the coding standards portion of the template according to the Module Three milestone requirements. In Project One, follow the instructions to add a layer of security to the existing coding standards. Please start each standard on a new page, as they may take up more than one page. The first seven coding standards are labeled by category. The last three are blank so you may choose three additional standards. Be sure to label them by category and give them a sequential number for that category. Add compliant and non-compliant sections as needed to each coding standard.

#### Coding Standard 1

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Type** | STD-001-CPP | Do not declare more than one variable per declaration |

| **Noncompliant Code** |
| --- |
| This example shows a misleading declaration where only one variable is initialized, which can cause confusion. |
| int a, b = 2;// Only ‘b’ is initialized, ‘a’ is uninitialized |

| **Compliant Code** |
| --- |
| This version ensures all variables are explicitly initialized, improving clarity. |
| int a = 2;  int b = 2; |

Source:   
SEI CERT. (n.d.). DCL04-C: Do not declare more than one variable per declaration. Carnegie Mellon University. Retrieved May 24, 2025, from <https://wiki.sei.cmu.edu/confluence/display/c/DCL04-C.+Do+not+declare+more+than+one+variable+per+declaration>

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

| **Principles(s):**  4: Keep It Simple - Declaring one variable per line improves readability and reduces logic errors.  9: Use Effective Quality Assurance Techniques - Simplifies static analysis and improves detection of bugs.  10: Adopt a Secure Coding Standard - Aligns with SEI CERT rules to promote secure, consistent coding. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/c/Axivion+Bauhaus+Suite) | 7.2.0 | CertC-DCL04 | - |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 9.0p0 | LANG.STRUCT.DECL.ML | Multiple Declarations on Line |
| [ECLAIR](https://wiki.sei.cmu.edu/confluence/display/c/ECLAIR) | 1.2 | CC2.DCL04 | Fully implemented |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/c/LDRA) | 9.7.1 | 579 S | Fully implemented |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | STD-002-CPP | Do not read uninitialized memory |

| **Noncompliant Code** |
| --- |
| This code attempts to return the value of an uninitialized local variable ‘x’. Accessing uninitialized memory results in undefined behavior and may lead to program instability or incorrect results. |
| int getValue() {  int x; // Uninitialized variable  return x; // Undefined behavior: reading uninitialized memory  } |

| **Compliant Code** |
| --- |
| This code initializes the variable ‘x’ to zero before it is used. Initializing variables before access ensures predictable behavior and eliminates the risk of undefined behavior. |
| int getValue() {  int x = 0; // Proper initialization  return x; // Safe to use  } |

Source:

SEI CERT. (n.d.). EXP33-C: Do not read uninitialized memory. Carnegie Mellon University. Retrieved May 24, 2025, from <https://wiki.sei.cmu.edu/confluence/display/c/EXP33-C.+Do+not+read+uninitialized+memory>

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

| **Principles(s):**  1: Validate Input Data - Ensures variables are initialized before use.  2: Heed Compiler Warnings - Compilers often warn about uninitialized variables.  9: Use Effective Quality Assurance Techniques - Static analysis and testing catch uninitialized use.  10: Adopt a Secure Coding Standard - Based on trusted CERT guidance. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Probable | Medium | P12 | L1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astree](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152428) | 24.04 | uninitialized-local-read  uninitialized-variable-use | Fully checked |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/c/Axivion+Bauhaus+Suite) | 7.2.0 | CertC-EXP33 | - |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 9.0p0 | LANG.MEM.UVAR | Uninitialized variable |
| [Coverity](https://wiki.sei.cmu.edu/confluence/display/c/Coverity) | 2017.07 | UNINIT | Implemented |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | STD-003-CPP | Guarantee that storage for strings has sufficient space for character data and the null terminator |

| **Noncompliant Code** |
| --- |
| This code reads input into a fixed-size character array using “std::cin >> buf;” without limiting the input size, which can lead to buffer overflow if the user enters more than 11 characters (excluding the null terminator). |
| char buf[12];  std::cin >> buf; //Risk of buffer overflow if input exceeds 12 characters |

| **Compliant Code** |
| --- |
| Using “std::string” for input eliminates the risk of overflow because the string automatically resizes and manages null termination. |
| std::string input;  std::cin >> input; // Using std::string avoids overflow |

Source:

SEI CERT. (n.d.). STR50-CPP: Guarantee that storage for strings has sufficient space for character data and the null terminator. Carnegie Mellon University. Retrieved May 24, 2025, from <https://wiki.sei.cmu.edu/confluence/display/cplusplus/STR50-CPP.+Guarantee+that+storage+for+strings+has+sufficient+space+for+character+data+and+the+null+terminator>

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

| **Principles(s):**  1: Validate Input Data - Verifying format strings prevents format injection.  7: Sanitize Data Sent to Other Systems - Sanitizing user input avoids malicious formatting.  9: Use Effective Quality Assurance Techniques -Testing can uncover dangerous format string usage. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astree](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) | 9.0p0 | MISC.MEM.NTERM  LANG.MEM.BO  LANG.MEM.TO | No space for null terminator  Buffer overrun  Type overrun |
| [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 |
| [RuleChecker](https://wiki.sei.cmu.edu/confluence/display/cplusplus/RuleChecker) | 22.10 | stream-input-char-array | Partially checked |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | STD-004-CCP | Exclude user input from format strings |

| **Noncompliant Code** |
| --- |
| This example uses user input directly as the format string in printf, which can lead to a format string vulnerability. If the user includes format specifiers (like %s or %x), it can expose memory or crash the program. |
| char userInput[100];  std::cin >> userInput;  printf(userInput); // Unsafe: User controls format string |

| **Compliant Code** |
| --- |
| This version defines the format string explicitly and passes the user input as a parameter. This approach ensures the format string is controlled and not influenced by the user. |
| char userInput[100];  std::cin >> userInput;  printf("%s", userInput); // Safe: Format string is fixed |

Source:

SEI CERT. (n.d.). FIO30-C: Exclude user input from format strings. Carnegie Mellon University. Retrieved May 24, 2025, from <https://wiki.sei.cmu.edu/confluence/display/c/FIO30-C.+Exclude+user+input+from+format+strings>

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

| **Principles(s):**  7: Sanitize Data Sent to Other Systems - Prevents injection attacks through unsafe SQL.  1: Validate Input Data - Input checks stop unsafe SQL queries.  5: Default Deny - Avoids excessive data exposure by limiting query access.  10: Adopt a Secure Coding Standard - Secure SQL practices are emphasized in standards. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/c/Axivion+Bauhaus+Suite) | 7.2.0 | CertC-FIO30 | Partially implemented |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 9.0p0 | IO.INJ.FMT  MISC.FMT | Format string injection  Format string |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/c/LDRA) | 9.7.1 | 86 D | Partially Implemented |
| [PC-lint Plus](https://wiki.sei.cmu.edu/confluence/display/c/PC-lint+Plus) | 1.4 | 592 | Partially supported: reports non-literal format strings |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | STD-005-CPP | Properly deallocate dynamically allocated resources |

| **Noncompliant Code** |
| --- |
| This code allocates memory using new but fails to call delete, causing a memory leak. If repeated, this could exhaust available memory and degrade system performance. |
| int\* ptr = new int[10]; //Memory is never freed |

| **Compliant Code** |
| --- |
| This version allocates memory using “new” and correctly deallocates it with “delete[]”, preventing memory leaks and maintaining proper memory hygiene. |
| int\* ptr = new int[10]; //use the memory  delete[] ptr; // Memory properly freed |

Source:

SEI CERT. (n.d.). MEM51-CPP: Properly deallocate dynamically allocated resources. Carnegie Mellon University. Retrieved May 24, 2025, from <https://wiki.sei.cmu.edu/confluence/display/cplusplus/MEM51-CPP.+Properly+deallocate+dynamically+allocated+resources>

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

| **Principles(s):**  8: Practice Defense in Depth - Memory protection helps prevent multiple forms of attack.  2: Heed Compiler Warnings - Many memory errors trigger compiler warnings.  9: Use Effective Quality Assurance Techniques - QA tools detect overflows and leaks.  10: Adopt a Secure Coding Standard - Covers memory handling techniques. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | 232 S, 236 S, 239 S, 407 S, 469 S, 470 S, 483 S, 484 S, 485 S, 64 D, 112 D | Partially implemented |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 9.0p0 | ALLOC.DF  ALLOC.TM  ALLOC.LEAK | Double free  Type mismatch  Leak |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Clang) | 3.9 | clang-analyzer-cplusplus.NewDeleteLeaks  -Wmismatched-new-delete  clang-analyzer-unix.MismatchedDeallocator | Checked by clang-tidy, but does not catch all violations of this rule |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2024b | CERT C++: MEM51-CPP | Checks for:  Invalid deletion of pointer  Invalid free of pointer  Deallocation of previously deallocated pointer  Rule partially covered. |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | STD-006-CPP | Detect and handle undefined behavior due to violations of program invariants |

| **Noncompliant Code** |
| --- |
| This code assumes that a divisor is nonzero without checking. Dividing by zero leads to undefined behavior and may crash the program. |
| int x = 10;  int y = 0;  int result = x / y; // Undefined behavior: division by zero |

| **Compliant Code** |
| --- |
| This code uses “assert()” to verify that the divisor is not zero before performing the division, enforcing a program invariant and preventing undefined behavior. |
| int x = 10;  int y = 2;  assert(y != 0); // Validates divisor before division  int result = x / y; |

Source:

SEI CERT. (n.d.). MSC12-C: Detect and remove code that has no effect or is never executed. Carnegie Mellon University. Retrieved May 24, 2025, from <https://wiki.sei.cmu.edu/confluence/display/c/MSC12-C.+Detect+and+remove+code+that+has+no+effect+or+is+never+executed>

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

| **Principles(s):**  9: Use Effective Quality Assurance Techniques - Assertions validate code assumptions during testing.  4: Keep It Simple - Clear, focused assertions enhance code clarity.  2: Heed Compiler Warnings - Compilers can detect unreachable or broken assertions.  10: Adopt a Secure Coding Standard - Encourages use of assertions when appropriate. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Unlikely | Low | P2 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Splint](https://wiki.sei.cmu.edu/confluence/display/c/Splint) | 3.1.1 | -standard | The default mode checks for unreachable code. |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/c/Polyspace+Bug+Finder) | R2024b | CERT C: Rec. MSC12-C | Checks for:   * Unreachable code * Dead code   Rec. partially covered. |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/c/LDRA) | 9.7.1 | 8 D, 65 D, 105 D, I J, 139 S, 140 S, 57 S | Partially implemented |
| [PVS-Studio](https://wiki.sei.cmu.edu/confluence/display/c/PVS-Studio) | 7.37 | V551, V606, V649, V779 | - |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | STD-007-CPP | Honor exception specifications |

| **Noncompliant Code** |
| --- |
| This code declares a function with “throw()”, meaning it should throw no exceptions, but then throws an exception. This behavior is misleading and results in “std::unexpected” being called. |
| void unsafe() throw() {  throw std::runtime\_error("Unexpected exception");  } |

| **Compliant Code** |
| --- |
| This version either avoids throwing exceptions entirely or correctly uses “noexcept” to indicate the function won’t throw, aligning with its specification. |
| void safe() noexcept {  // No exceptions are thrown  } |

Source:

SEI CERT. (n.d.). ERR55-CPP: Honor exception specifications. Carnegie Mellon University. Retrieved May 24, 2025, from <https://wiki.sei.cmu.edu/confluence/display/cplusplus/ERR55-CPP.+Honor+exception+specifications>

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

| **Principles(s):**  4: Keep It Simple - Clear and consistent exception handling prevents cascading errors.  2: Heed Compiler Warnings - Compilers warn on unhandled or unsafe exceptions.  9: Use Effective Quality Assurance Techniques - Testing reveals improper exception flows.  10: Adopt a Secure Coding Standard - Ensures correct exception usage. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | unhandled-throw-noexcept | Partially checked |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 9.0p0 | LANG.STRUCT.EXCP.THROW | Use of throw |
| [Parasoft C/C++Test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2024.2 | CERT\_CPP-ERR55-a | Where a function's declaration includes an exception-specification, the function shall only be capable of throwing exceptions of the indicated type(s) |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2024b | CERT C++: ERR55-CPP | Checks for noexcept functions exiting with exception (rule fully covered) |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Memory | STD-008-CPP | Do not access freed memory |

| **Noncompliant Code** |
| --- |
| This code deallocates memory with “delete” and then attempts to access it. Accessing freed memory results in undefined behavior. |
| int\* ptr = new int(42);  delete ptr;  int value = \*ptr; //Undefined behavior: ptr is a dangling pointer |

| **Compliant Code** |
| --- |
| This version sets the pointer to “nullptr” immediately after deallocation. This prevents accidental reuse of freed memory. |
| int\* ptr = new int(42);  delete ptr;  ptr = nullptr; //Prevent dangling pointer |

Source:  
SEI CERT. (n.d.). MEM50-CPP: Do not access freed memory. Carnegie Mellon University. Retrieved May 24, 2025, from <https://wiki.sei.cmu.edu/confluence/display/cplusplus/MEM50-CPP.+Do+not+access+freed+memory>

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

| **Principles(s):**  6: Adhere to the Principle of Least Privilege - Authorization checks restrict access.  5: Default Deny - Only allow explicitly approved permissions.  3: Architect and Design for Security Policies - Authorization is a design-level concern.  10: Adopt a Secure Coding Standard - Reinforces secure access control. |
| --- |

**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 | dangling\_pointer\_use | - |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 9.0p0 | ALLOC.UAF | Use after free |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2024.2 | CERT\_CPP-MEM50-a | Do not use resources that have been freed |
| [Coverity](https://wiki.sei.cmu.edu/confluence/display/c/Coverity) | v7.5.0 | USE\_AFTER\_FREE | Can detect the specific instances where memory is deallocated more than once or read/written to the target of a freed pointer |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Strings | STD-009-CPP | Ensure that all memory is initialized before use |

| **Noncompliant Code** |
| --- |
| This code attempts to create a “std::string” from a null pointer, which results in undefined behavior and typically crashes the program. |
| const char\* ptr = nullptr;  std::string str(ptr); //Undefined behavior: null pointer used |

| **Compliant Code** |
| --- |
| This version checks if the pointer is null before using it to construct a “std::string”, ensuring program stability. |
| const char\* ptr = nullptr;  std::string str = (ptr != nullptr) ?  std::string(ptr) : std::string("default"); |

Source:

SEI CERT. (n.d.). STR51-CPP: Do not attempt to create a std::string from a null pointer. Carnegie Mellon University. Retrieved May 24, 2025, from <https://wiki.sei.cmu.edu/confluence/display/cplusplus/STR51-CPP.+Do+not+attempt+to+create+a+std%3A%3Astring+from+a+null+pointer>

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

| **Principles(s):**  3: Architect and Design for Security Policies - Authentication must be considered in system design.  6: Adhere to the Principle of Least Privilege - Authenticated roles define privilege.  5: Default Deny - Block unauthenticated access by default.  10: Adopt a Secure Coding Standard - Encourages proper authentication mechanisms. |
| --- |

**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 | assert\_failure | - |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 9.0p0 | LANG.MEM.NPD | Null Pointer Dereference |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2024.2 | CERT\_CPP-STR51-a | Avoid null pointer dereferencing |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/c/Polyspace+Bug+Finder) | R2024b | CERT C++: STR51-CPP | Checks for string operations on null pointer (rule partially covered). |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Declaration | STD-010-CPP | Overload allocation and deallocation functions as a pair in the same scope |

| **Noncompliant Code** |
| --- |
| This code overloads “operator new” but forgets to overload “operator delete”. This can result in mismatched memory management, leading to resource leaks or undefined behavior. |
| class MyClass {  public:  void\* operator new(std::size\_t size); //Missing: operator delete  }; |

| **Compliant Code** |
| --- |
| This version correctly overloads both “operator new” and “operator delete”, ensuring consistent memory allocation and deallocation. |
| class MyClass {  public:  void\* operator new(std::size\_t size);  void operator delete(void\* ptr);  }; |

Source:

SEI CERT. (n.d.). DCL54-CPP: Overload allocation and deallocation functions as a pair in the same scope. Carnegie Mellon University. Retrieved May 24, 2025, from <https://wiki.sei.cmu.edu/confluence/display/cplusplus/DCL54-CPP.+Overload+allocation+and+deallocation+functions+as+a+pair+in+the+same+scope>

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

| **Principles(s):**  8: Practice Defense in Depth - Auditing logs add additional security layers.  9: Use Effective Quality Assurance Techniques - Logging helps trace errors.  6: Adhere to the Principle of Least Privilege - Logs can audit privilege abuse.  10: Adopt a Secure Coding Standard - Defines safe and reliable audit logging. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | new-delete-pairwise | Partially checked |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Clang) | 3.9 | misc-new-delete-overloads | Checked with clang-tidy. |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2024.2 | CERT\_CPP-DCL54-a | Always provide new and delete together |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2024b | CERT C++: DCL54-CPP | Checks for mismatch between overloaded operator new and operator delete (rule fully covered) |

### 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.

**Automation Summary:**

Automation will be integrated throughout the DevSecOps pipeline to ensure early detection and enforcement of secure coding standards. At the Plan and Create stages, IDE plugins and code templates will guide developers in following secure patterns. During Verify, tools such as SonarQube, Cppcheck, and Clang-Tidy will perform static analysis to identify issues before the build. Pre-Production testing will simulate input fuzzing and runtime problems. At Release, automated software signing guarantees package integrity. In the Prevent and Detect stages, monitoring tools and runtime protections like RASP will identify violations. All findings will contribute to incident response during the Respond and Adapt stages to maintain compliance. This strategy promotes both shift-left security and audit readiness.

### 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 | Low | P3 | L3 |
| STD-002-CPP | High | Probable | Medium | P12 | L1 |
| STD-003-CPP | High | Likely | Medium | P18 | L1 |
| STD-004-CPP | High | Likely | Medium | P18 | L1 |
| STD-005-CPP | High | Likely | Low | P18 | L1 |
| STD-006-CPP | Low | Unlikely | Low | P2 | L3 |
| STD-007-CPP | Low | Likely | Medium | P9 | L2 |
| STD-008-CPP | High | Likely | Medium | P18 | L1 |
| STD-009-CPP | High | Likely | Low | P18 | L1 |
| STD-010-CPP | Low | Probable | Low | P6 | L2 |

### 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 | This pertains to safeguarding stored data, including files or databases, by employing robust encryption algorithms. This guarantees that sensitive information remains unreachable if the physical or digital storage is compromised. |
| Encryption in flight | Data transmitted over networks must be encrypted using protocols like TLS or SSL. This protects data from interception, tampering, and eavesdropping while it is moving between systems. |
| Encryption in use | Data should be protected while being processed, such as by using secure memory allocation or hardware-based secure enclaves. This policy defends against runtime memory inspection and data leaks during operation. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | All users must verify their identity through secure authentication methods. This ensures that system access is restricted to verified individuals and minimizes the risk of unauthorized use. |
| Authorization | Users must be granted access rights based on their role or responsibility. Access should be restricted to the minimum necessary to perform assigned duties, following the principle of least privilege. |
| Accounting | It is essential to log and monitor all user and system activities. These audit logs are critical for detecting incidents, ensuring accountability, and supporting forensic investigations in the event of a security breach. |

**\***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 |
| --- | --- | --- | --- | --- |
| 2.0 | 06/18/2025 | Final Template | Courtney Warner |  |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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