**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 | Validating input data means putting up guardrails around what users can submit into our system. By implementing validation, the organization is protecting itself from the vast majority of software vulnerabilities. |
| 1. Heed Compiler Warnings | When we compile our code, we should use the highest warning level available and fix any warnings that arise. This practice, along with the use of other analysis tools, will reduce our security flaws. |
| 1. Architect and Design for Security Policies | Security considerations should start in the earliest stages of the project before code is even written. We need to ensure any solution includes a way to implement and enforce our security policies. |
| 1. Keep It Simple | Simple systems are easier to protect because as solutions grow in complexity, so does the chance for mistakes to occur. Our designs should strive to be as simple as possible. |
| 1. Default Deny | All access to our systems should be given, not revoked. In other words, we should automatically deny entry to anyone who has not previously requested and been granted permissions. |
| 1. Adhere to the Principle of Least Privilege | Our applications should use the lowest level of permissions possible when executing processes. If elevated permissions are needed, they should only be accessed when that process is happening. Using elevated permissions as little as possible reduces the risk of an attacker gaining access to those permissions. |
| 1. Sanitize Data Sent to Other Systems | Just as we validate any data coming into our systems, we need to sanitize data going out to complex subsystems. Because the subsystems are not aware of the context of the call, it is the responsibility of the sender to ensure the data is cleaned before it make the call. |
| 1. Practice Defense in Depth | It is not enough to employ one security solution. We need to have layers of protection to ensure vulnerabilities do not become incidents. |
| 1. Use Effective Quality Assurance Techniques | Quality can play an important role in security. Fuzz testing, pen testing, and code audits can help identify vulnerabilities before they are introduced into production. |
| 1. Adopt a Secure Coding Standard | We must develop and distribute a security standard for our organization. |

### 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.

A white square with yellow text

AI-generated content may be incorrect.

**MEM50-CPP**

**DCL57-CPP**

**STR51-CPP**

**ERR51-CPP**

**MEM52-CPP**

**MEM51-CPP**

**STR02-C**

**STR53-CPP**

**MSC50-CPP**

**DCL52-CPP**

#### Coding Standard 1

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Type** | DCL52-CPP | Never qualify a reference type with const or volatile  Rationale: C++ prohibits or ignores when variables are qualified with const or volatile. This issue with continuing to use these qualifiers is that they may result in undefined behavior that may expose vulnerabilities. |

| **Noncompliant Code** |
| --- |
| The code below shows the char p being assigned as a const, but then reassigned on the next line. |
| #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 compliant code below does not qualify p as a const, making the reassignment on the next line acceptable. |
| #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):**   * Heed compiler warnings: the compiler should warn about this issue * Use Effective Quality Assurance Techniques: The below automation would help detect this issue * Adopt a Secure Coding Standard: Because this is part of the C++ Secure Coding Standard, we are aligning our standard to industry best-practices |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2024.2 | **CERT\_CPP-DCL52-a** | Never qualify a reference type with 'const' or 'volatile' |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2024b | [CERT C++: DCL52-CPP](https://www.mathworks.com/help/bugfinder/ref/certcdcl52cpp.html) | Checks for:   * const-qualified reference types * Modification of const-qualified reference types   Rule fully covered. |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/cplusplus/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](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046388) | 4.10 | [**S3708**](https://www.sonarsource.com/products/codeanalyzers/sonarcfamilyforcpp/rules-cpp.html#RSPEC-3708) |  |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | MSC50-CPP | Do not use std::rand() for generating pseudorandom numbers  Rationale: The use of std::rand() should not be used for processes requiring random numbers because they can be predictable and insecure. Especially if needed for security purposes like encryption, a different library should be used that is known to be secure. |

| **Noncompliant Code** |
| --- |
| The below code snipper is incorrect because it uses std::rand to generate an ID. These ids would be predictable. |
| #include <cstdlib>  #include <string>    void f() {  std::string id("ID"); // Holds the ID, starting with the characters "ID" followed by a random integer in the range [0-10000].  id += std::to\_string(std::rand() % 10000);  // ...  } |

| **Compliant Code** |
| --- |
| Instead of using std::rand, this code block uses the more robust std::mt19937 for random number generation. |
| #include <random>  #include <string>    void f() {  std::string id("ID"); // Holds the ID, starting with the characters "ID" followed by a random integer in the range [0-10000].  std::uniform\_int\_distribution<int> distribution(0, 10000);  std::random\_device rd;  std::mt19937 engine(rd());  id += std::to\_string(distribution(engine));  // ...  } |

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

| **Principles(s):**   * Validate Input Data: Though this rule does not deal with user input, the input the system is generating also needs validation. This rule helps clean this input up. * Practice Defense in Depth: Because random numbers are often used in authentication and cryptography, ensuring these numbers are string is an important part of our DID. * Use Effective Quality Assurance Techniques: The below automation would help detect this issue * Adopt a Secure Coding Standard: Because this is part of the C++ Secure Coding Standard, we are aligning our standard to industry best-practices |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2024.2 | **CERT\_CPP-MSC50-a** | Do not use the rand() function for generating pseudorandom numbers |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2024b | [CERT C++: MSC50-CPP](https://www.mathworks.com/help/bugfinder/ref/certcmsc50cpp.html) | Checks for use of vulnerable pseudo-random number generator (rule partially covered) |
| [RuleChecker](https://wiki.sei.cmu.edu/confluence/display/cplusplus/RuleChecker) | 22.10 | **bad-function (AUTOSAR.26.5.1A)** | Fully checked |
| [Security Reviewer - Static Reviewer](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Security+Reviewer+-+Static+Reviewer) | 6.02 | **RTOS\_07** | Fully implemented |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | STR53-CPP | Range check element access  Rationale: When access string elements using operator, you must ensure the index is not out of range. Similarly, if using .front() or .back(), you must make sure the string is not empty. If either of these situations occur, there will be no error and they will result in undefined behavior. |

| **Noncompliant Code** |
| --- |
| The issue with the below code is that the value returned by the call to get\_index() may be grater than the size of the string. |
| #include <string>    **extern** std::**size\_t** get\_index();    **void** f() {    std::string s("01234567");    s[get\_index()] = '1';  } |

| **Compliant Code** |
| --- |
| Here we employ a try/catch block to protect against this scenario. |
| #include <stdexcept>  #include <string>  **extern** std::**size\_t** get\_index();    **void** f() {    std::string s("01234567");  **try** {      s.at(get\_index()) = '1';    } **catch** (std::out\_of\_range &) {      // Handle error    }  } |

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

| **Principles(s):**   * Validate Input Data: This standard validates the index passed into the function is valid * Keep It Simple: using at() is the simplest way to implement sting index operators * Use Effective Quality Assurance Techniques: The below automation would help detect this issue * Adopt a Secure Coding Standard: Because this is part of the C++ Secure Coding Standard, we are aligning our standard to industry best-practices |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 9.1p0 | **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 |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2025.2 | **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. |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | STR02-C | Sanitize data passed to complex subsystems  Rationale: It is required to make sure all data that we pass to complex subsystems, like SQL, are sanitized before they are passed. Providing offending strings that can trigger commands or actions is a common type of attack. Sanitizing our data will protect us from that vulnerability. |

| **Noncompliant Code** |
| --- |
| In the below example there is no sanitizing of the data, leaving a pretty big vulnerability open. |
| **sprintf**(buffer, "/bin/mail %s < /tmp/email", addr);  **system**(buffer); |

| **Compliant Code** |
| --- |
| The below approach is used to sanitized the data. Here we set a list of approved characters and loop through the string to ensure nothing else gets through. |
| **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):**   * Validate Input Data: This standard ensures we only pass safe and clean data to complex subsystems * Sanitize Data Sent to Other Systems: This standard ensures we only pass safe and clean data to complex subsystems * Use Effective Quality Assurance Techniques: The below automation would help detect this issue * Adopt a Secure Coding Standard: Because this is part of the C++ Secure Coding Standard, we are aligning our standard to industry best-practices |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Klocwork](https://wiki.sei.cmu.edu/confluence/display/c/Klocwork) | 2025.2 | **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) | 2024.2 | **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) | R2024b | [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** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | MEM51-CPP | Properly deallocate dynamically allocated resources  Rationale: This rule states that any dynamically allocated memory must be deallocated correctly and in a timely manner to help prevent memory leaks. |

| **Noncompliant Code** |
| --- |
| In the below example the developer did try to deallocate the memory held by the array, but they used a scalar delete instead of an array delete, resulting in undefined behavior. |
| **void** f() {  **int** \*array = **new** **int**[10];    // ...  **delete** array;  } |

| **Compliant Code** |
| --- |
| This example uses the correct deallocation function. |
| **void** f() {  **int** \*array = **new** **int**[10];    // ...  **delete**[] array;  } |

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

| **Principles(s):**   * Architect and Design for Security: Memory management is an important part of the initial coding design * Use Effective Quality Assurance Techniques: The below automation would help detect this issue * Adopt a Secure Coding Standard: Because this is part of the C++ Secure Coding Standard, we are aligning our standard to industry best-practices |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [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++: MEM51-CPP](https://www.mathworks.com/help/bugfinder/ref/certcmem51cpp.html) | Checks for:   * Invalid deletion of pointer * Invalid free of pointer * Deallocation of previously deallocated pointer   Rule partially covered. |
| [PVS-Studio](https://wiki.sei.cmu.edu/confluence/display/cplusplus/PVS-Studio) | 7.38 | [**V515**](https://pvs-studio.com/en/docs/warnings/v515/), [**V554**](https://pvs-studio.com/en/docs/warnings/v554/), [**V611**](https://pvs-studio.com/en/docs/warnings/v611/), [**V701**](https://pvs-studio.com/en/docs/warnings/v701/), [**V748**](https://pvs-studio.com/en/docs/warnings/v748/), [**V773**](https://pvs-studio.com/en/docs/warnings/v773/), [**V1066**](https://pvs-studio.com/en/docs/warnings/v1066/) |  |
| [Security Reviewer - Static Reviewer](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Security+Reviewer+-+Static+Reviewer) | 6.02 | **wcsdupCalled** | Fully implemented |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | MEM52-CPP | Detect and handle memory allocation errors  Rationale: Dynamic memory allocations must be checked for failures. If the non-throwing form of new is used, the return should be tested against nullptr. If using the throwing form, you don’t have to check explicitly because it will fire an exception, but you do have to implement logic to handle that exception. |

| **Noncompliant Code** |
| --- |
| In the below code, the throwing form of new is used, but the function is marked noexcept. If the function fails, unexpected behavior can occur. |
| #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** |
| --- |
| In the below code, the throwing form is used the function is explicitly marked as noexcept(false) so the caller will no that exceptions are a possibility. |
| #include <cstring>    **void** f(**const** **int** \*array, std::**size\_t** size) noexcept(**false**) {  **int** \*copy = **new** **int**[size];    // If the allocation fails, it will throw an exception which the caller    // will have to handle.    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):**   * Architect and Design for Security: Memory management is an important part of the initial coding design * Keep it Simple: This standard expressly states the safest and easiest way to implement dynamic memory allocations * Use Effective Quality Assurance Techniques: The below automation would help detect this issue * Adopt a Secure Coding Standard: Because this is part of the C++ Secure Coding Standard, we are aligning our standard to industry best-practices |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | Medium | P27 | L1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [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) |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | ERR51-CPP | Handle all exceptions  Rationale: When an exception is thrown, the code tries to find a matching handler to assign the exception to. If it cannot find a matching handler, it will abnormally terminate, which can cause destructors to not be called. This type of behavior makes us vulnerable to denial-of-service attacks, which is why it is critical that all exceptions have handlers. |

| **Noncompliant Code** |
| --- |
| In this code example there is no catch block and there are no defined handlers if throwing\_func does return with an exception. This would cause an abnormal termination. |
| **void** throwing\_func() noexcept(**false**);    **void** f() {    throwing\_func();  }    **int** main() {    f();  } |

| **Compliant Code** |
| --- |
| Here we the developer has implemented a try/catch block and has guaranteed the stack will unwind up to main(). |
| **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: Exception handling should first be discussed in the design phase to ensure no handlers are missed * Keep it Simple: Better failures help set us up for quicker fixes * Use Effective Quality Assurance Techniques: The below automation would help detect this issue * Adopt a Secure Coding Standard: Because this is part of the C++ Secure Coding Standard, we are aligning our standard to industry best-practices |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [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 |
| [Security Reviewer - Static Reviewer](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Security+Reviewer+-+Static+Reviewer) | 6.02 | **C35** | Fully implemented |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Characters and Strings | STR51-CPP | Do not attempt to create a std::string from a null pointer  Rationale: Passing a null pointer to a std::basic\_string operation will result in undefined behavior. |

| **Noncompliant Code** |
| --- |
| In the below code a string object is created from the getenv() function. If that function returns a null pointer, this will result in undefined behavior. |
| #include <cstdlib>  #include <string>    **void** f() {    std::string tmp(std::**getenv**("TMP"));  **if** (!tmp.empty()) {      // ...    }  } |

| **Compliant Code** |
| --- |
| Here the developer verifies getenv() will not return null before assigning. |
| #include <cstdlib>  #include <string>    **void** f() {  **const** **char** \*tmpPtrVal = std::**getenv**("TMP");    std::string tmp(tmpPtrVal ? tmpPtrVal : "");  **if** (!tmp.empty()) {      // ...    }  } |

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

| **Principles(s):** [Name the principle and explain how it maps to this standard.]   * Validate Input Data: This standard ensures invalid data is not used to initiate a string * Keep it simple: Applies a simple coding best practice for implementing strings * Use Effective Quality Assurance Techniques: The below automation would help detect this issue * Adopt a Secure Coding Standard: Because this is part of the C++ Secure Coding Standard, we are aligning our standard to industry best-practices |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | Medium | P18 | L2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 9.1p0 | **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](https://www.mathworks.com/help/bugfinder/ref/certcstr51cpp.html) | Checks for string operations on null pointer (rule partially covered). |
| [Security Reviewer - Static Reviewer](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Security+Reviewer+-+Static+Reviewer) | 6.02 | **shiftTooManyBits** | Fully implemented |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Declarations and Initializations | DCL57-CPP | Don’t let exceptions escape from destructors or deallocation functions  Rationale: Functions that deallocate memory or use destructors should be marked as noexcept because throwing exceptions from these types of functions can result in undefined behavior. |

| **Noncompliant Code** |
| --- |
| Here is an example of a global deallocation function that has been marked as noexcept(false). If any exceptions are returned from this function, undefined behavior will occur. |
| #include <stdexcept>    **bool** perform\_dealloc(**void** \*);    **void** operator **delete**(**void** \*ptr) noexcept(**false**) {  **if** (perform\_dealloc(ptr)) {  **throw** std::logic\_error("Something bad");    }  } |

| **Compliant Code** |
| --- |
| Here is an example of the same function, but it does not fire exceptions and fails as gracefully as possible. |
| #include <cstdlib>  #include <stdexcept>    **bool** perform\_dealloc(**void** \*);  **void** log\_failure(**const** **char** \*);    **void** operator **delete**(**void** \*ptr) noexcept(**true**) {  **if** (perform\_dealloc(ptr)) {      log\_failure("Deallocation of pointer failed");      std::**exit**(1); // Fail, but still call destructors    }  } |

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

| **Principles(s):**   * Architect and Design for Security: Exception handling should first be discussed in the design phase * Keep It Simple: The standard dictates clean destructors in the code * Use Effective Quality Assurance Techniques: The below automation would help detect this issue * Adopt a Secure Coding Standard: Because this is part of the C++ Secure Coding Standard, we are aligning our standard to industry best-practices |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2024.2 | **CERT\_CPP-DCL57-a** **CERT\_CPP-DCL57-b** | Never allow an exception to be thrown from a destructor, deallocation, and swap Always catch exceptions |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2024b | [CERT C++: DCL57-CPP](https://www.mathworks.com/help/bugfinder/ref/certcdcl57cpp.html) | Checks for class destructors exiting with an exception (rule partially covered) |
| [PVS-Studio](https://wiki.sei.cmu.edu/confluence/display/cplusplus/PVS-Studio) | 7.38 | [**V509**](https://pvs-studio.com/en/docs/warnings/v509/)**,**[**V1045**](https://pvs-studio.com/en/docs/warnings/v1045/) |  |
| [RuleChecker](https://wiki.sei.cmu.edu/confluence/display/cplusplus/RuleChecker) | 22.10 | **destructor-without-noexcept** **delete-without-noexcept** | Fully checked |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Memory management | MEM50-CPP | Do not access freed memory  Rationale: once memory has been deallocated, the pointer referring to it becomes a dangling point. Evaluating that pointer will result in undefined behavior. |

| **Noncompliant Code** |
| --- |
| In the below example s is referenced after being deallocated. This creates a vulnerability that could be made to run attacker-introduced code with the permissions of the deleted process. |
| #include <new>    **struct** S {  **void** f();  };    **void** g() noexcept(**false**) {    S \*s = **new** S;    // ...  **delete** s;    // ...    s->f();  } |

| **Compliant Code** |
| --- |
| In this example s is not deleted until it is truly no longer needed. |
| #include <new>    **struct** S {  **void** f();  };    **void** g() noexcept(**false**) {    S \*s = **new** S;    // ...    s->f();  **delete** s;  } |

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

| **Principles(s):**   * Architect and Design for Security: Memory management is an important part of the initial coding design * Keep it Simple: This standard dictates how to handle freed memory in the cleanest way * Default Deny: Keeps memory that has been freed from being accessed * Use Effective Quality Assurance Techniques: The below automation would help detect this issue * Adopt a Secure Coding Standard: Because this is part of the C++ Secure Coding Standard, we are aligning our standard to industry best-practices |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [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++: MEM50-CPP](https://www.mathworks.com/help/bugfinder/ref/certcmem50cpp.html) | Checks for:   * Pointer access out of bounds * Deallocation of previously deallocated pointer * Use of previously freed pointer   Rule partially covered. |
| [PVS-Studio](https://wiki.sei.cmu.edu/confluence/display/cplusplus/PVS-Studio) | 7.38 | [**V586**](https://pvs-studio.com/en/docs/warnings/v586/), [**V774**](https://pvs-studio.com/en/docs/warnings/v774/) |  |
| [Security Reviewer - Static Reviewer](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Security+Reviewer+-+Static+Reviewer) | 6.02 | **CPP\_12 CPP\_14 CPP\_15** | Fully implemen |

### 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 is an important part of our security and can be implemented at every phase of the DevSecOps flow. Even before the code is written, we should be planning and designing our automation solutions. The first two phases should include consideration for automation, the tools we’ll use, and the specific threats we may need to address based on our known coding solution.

Once we start building, we can begin automating at the local level with security-focused unit tests and dependency checks. The brunt of the automation will occur in the verify and test phase when the code has been deployed to a test environment. In this phase we can automate important vulnerability scans, compliance, and security testing. To have a successful automation strategy, we must ensure that the automation we build is enabled in the CI pipeline. This will ensure that we are verifying the security of all environments the code is deployed to. Once the code is deployed to the final production environment, we can also initiate automated penetration testing, another form of protection.

In the monitor and detect phase we will continue to run our previously built automation and use other tools, like logging detection, to catch any unexpected behavior early. This will set us up to be able to respond quickly if an alert is made so that we can maintain our service and our security.

### 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 |
| --- | --- | --- | --- | --- | --- |
| DCL52-CPP | Low | Unlikely | Low | P3 | L3 |
| DCL57-CPP | Low | Likely | Low | pg | L2 |
| ERR51-CPP | Low | Probable | Medium | P6 | L2 |
| MEM50-CPP | High | Likely | High | pg | L2 |
| MEM51-CPP | High | Likely | Medium | pg | L2 |
| MEM52-CPP | High | Likely | Medium | P27 | L1 |
| MSC50-CPP | Medium | Unlikely | Low | P4 | L3 |
| STR02-C | High | Liekly | Medium | pg | L2 |
| STR51-CPP | High | Likely | Medium | P18 | L2 |
| STR53-CPP | High | Unlikely | Low | P3 | L3 |

### 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 | The practice of encrypting data that is currently being stored and is not in use. This can include data on storage devices or in the cloud. The most common type of encryption for data at rest is symmetric cryptography, which uses the same key to encrypt and decrypt the data. This policy helps ensure that our highly important data is not at risk when it is not actively being used. |
| Encryption in flight | The practice of encrypting data as it is being transmitted from one entity to another. This includes common activities that happen everyday such as emailing documents, uploading or downloading data from a browser, or uploading from local storage to a cloud environment. Though we have ensured the data is safe at rest, that protection does not apply once the data is being transmitted. We must have policies in place that will protect against in-transit data ending up in the wrong hands, either through human error or malicious attack. If the data is encrypted before the transmission, the third-party recipient will not be able to consume any of it. |
| Encryption in use | The practice of encrypting data that is currently been read or processed. This type of encryption can have a heavy performance cost because it requires the system to encrypt and decrypt the data in real time as its doing its other processing work. We should be cognizant of the operation cost when we implement this type of security, but it is vital for any important data. The practice of encryption in use happens in a private region of main memory which also holds the keys. This allows the system to process and read the data at the same time as the encryption process, ensuring the real data is never exposed. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Is the practice of ensuring each user that accesses the system has a unique login that must be verified before they are granted access. We should implement a multi-factor authentication process that will use two prompts to ensure users are who they say they are. |
| Authorization | Is a level above authentication, which verifies a user’s identity. Authorization verifies each user has access to do only what is necessary for their job duties. For instance, a developer does not need access to the financial systems the company uses. Additionally, authorization ensures that even if a user has access, their possible impact can be controlled. For instance, not everyone who accesses a database needs to be able to write to that database or implement administrative changes to it so we should lock down that functionality for that user. Implementing an authorization hierarchy ensures users are only granted access to the parts of the system they need with the lowest level of capability possible. |
| Accounting | The final leg of the Triple-A framework is accounting, which allows us to understand how users are spending their time in our system. This could include logging information like how long a user was in the system, the data they viewed or changed, and even their IP address. Having this data allows us to monitor the activity on our system so we can spot potential risks or threats earlier. |

**\***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 | 7/20/2025 | Module Three Assignment | Marye Bierbaum |  |
| 1.2 | 8/17/2025 | Project One | Marye Bierbaum |  |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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