**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 | All external input should be treated as unsafe until proved to be safe. Proper validation and sanitization help prevent common attacks like SQL injections and buffer overflows. By enforcing strict checks on data type, length, and format, the system can better ensure that the data is safe to process. |
| 1. Heed Compiler Warnings | Compilers can offer important warnings and indication into unsafe code practices, type mismatches and deprecated functions, but still compile regardless. This means we need to heed these warnings that appear even if the code itself works. Ignoring these issues will lead to long-term issues, and potential bugs further down the line. |
| 1. Architect and Design for Security Policies | Security should be built into the foundation of the system, rather than being an afterthought. By aligning our goals with a security mindset as we proceed, we can ensure our systems are built around proper authentication, encryption, and authorization mechanisms from the start. A proactive approach to safety and security save time and money, as well as instilling a secure coding culture. |
| 1. Keep It Simple | Complexity can often be the enemy of security. Overly intricate or convoluted code creates more opportunities for mistakes and is harder to trouble shoot. By adhering to simplicity and clarity in the code, we reduce the chance of one Achillies heal spot taking down a whole system. |
| 1. Default Deny | A secure system should assume denial first, unless permissions are explicitly granted. The purpose of this principle is the same as creating a large wall with a few scattered doors, versus having an open expanse with only select check points. By starting from a restrictive baseline and selectively enabling necessary functionality, we can significantly reduce unauthorized access. |
| 1. Adhere to the Principle of Least Privilege | This principle is about granting the minimum amount of permission required for a user or function to perform their required function. Limiting privileges reduces the potential for accidently damages due to misconfiguration or bugs, and also limits how much damage a bad actor can get into should they gain access to that system, or use credentials. |
| 1. Sanitize Data Sent to Other Systems | Just as input needs to be sanitized, we must also sanitize output heading to other systems or functions, this will help minimize the damage of accident propagation of malicious code, should it be introduced somehow. Much like an air lock on a ship, we need to ensure we are protected both entering and leaving. |
| 1. Practice Defense in Depth | Security should not rely on a single mechanism, and instead multiple processes should be utilized to ensure as much DiD is used. This layered approach ensures that should one defense fail others remain in place to slow down attackers. |
| 1. Use Effective Quality Assurance Techniques | Using effective quality control techniques like automated analysis tools, code reviews, penetration testing, and others all come together to ensure a quality product in the end. By putting the software and system through the paces we can see things we wouldn’t see through just normal compiling and bug testing. |
| 1. Adopt a Secure Coding Standard | By following established secure coding standards, we can help create a culture of code safety within an organization. Staying rigid about what is and is not acceptable keeps us from taking shortcuts for speed and time and helps to stop preventable issues. As a team that holds one another accountable to a Secure coding standard, we can produce more resilient and trustworthy software. |

### C/C++ Ten Coding Standards

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

#### Coding Standard 1

| **Coding Standard** | **Label** | **Never qualify a reference type with const or volatile** |
| --- | --- | --- |
| **Data Type** | [STD-001-CPP] | References in C++ are already ‘const’ with respect to binding, once a reference is bound to an object you can’t make it refer to something else. Therefore, trying to do so could cause unexpected behaviors. |

| **Noncompliant Code** |
| --- |
| In this noncompliant example we are attempting to const qualify a char p as part of a declaration that uses a reference type. When we accidently write this code, it does not do the intended, and instead can cause undefined behavior. |
| Char &const p; |

| **Compliant Code** |
| --- |
| To be compliant and not risk undefined behavior we must write this process as: |
| char const &p;  //or use const char &p; |

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

| **Principles(s):** Adopt a Secure Coding Standard. This principle maps to this coding standard because it is taking secure coding into account as the code is developed. By preventing small accidents like this standard we can save time course correcting later. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Unlikely | Low (Would just need refactor) | P3 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Klocwork](https://www.securecoding.cert.org/confluence/display/cplusplus/Klocwork) | 2025.2 | **CERT.DCL.REF\_TYPE.CONST\_OR\_VOLATILE** |  |
| [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) | R2025b | [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. |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Do not cast to an out of range enumeration value** |
| --- | --- | --- |
| **Data Value** | [STD-002-CPP] | In C++, both scoped and unscoped enumerations can be set manually to represent values beyond those explicitly declared in their enumerators. Casting an integer or floating-point value to an enumeration type is only safe if the value falls within the enumeration’s representable range, otherwise the behavior is undefined. To prevent errors, programs should always validate values before casting them to an enumeration. |

| **Noncompliant Code** |
| --- |
| In this example of non-compliant code, the enumerator is being cast before being checked for being an out of range value, resulting in unspecified behavior. |
| enum EnumType {    One,  Two,  Three  };  void function (int variable) {    EnumType enumVariable = static\_cast<EnumType>(variable);      if (enumVariable < One || enumVariable > Three) {      // Handle the exception    }  } |

| **Compliant Code** |
| --- |
| In this compliant code example, the value being passed into function is checked whether it is out of range first, before it is cast to the enumerator, thus insuring an absence of undefined behaviour. |
| enum EnumType {    One,  Two,  Three  };  void function (int variable) {  if (variable < One || variable > Three) {       // Handle the exception     }  EnumType enumVariable = static\_cast<EnumType>(variable);  } |

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

| **Principles(s):** ValidateInput Data. This principle maps to this standard because the incoming variable must be checked or risk receiving out of bounds data. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Medium | Unlikely | Medium(Would just need validation added) | P4 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | **cast-integer-to-enum** | Partially checked |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 9.1p0 | **LANG.CAST.COERCE**  **LANG.CAST.VALUE** | Coercion Alters Value  Cast Alters Value |
| [RuleChecker](https://wiki.sei.cmu.edu/confluence/display/cplusplus/RuleChecker) | 22.10 | **cast-integer-to-enum** | Partially checked |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2025b | [CERT C++: INT50-CPP](https://www.mathworks.com/help/bugfinder/ref/certcint50cpp.html) | Checks for casting to out-of-range enumeration value (rule fully covered) |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Guarantee that storage for strings has sufficient space for character data and the null terminator.** |
| --- | --- | --- |
| **String Correctness** | [STD-003-CPP] | When copying data to a buffer that is not large enough to hold the data, it will result in a buffer overflow. This error, while a frequent concern, can easily be handled through truncation or ensuring the buffer is of sufficient size. Another important thing to remember that C-Style strings require a null character while C++ strings don’t. |

| **Noncompliant Code** |
| --- |
| This non-compliant code example utilizes a C-Style string to hold input, however because we must specify the size and include a null character, the user may enter more characters and cause a buffer overflow. |
| void function(){  char buffer[20];  std::cin >> buffer; } |

| **Compliant Code** |
| --- |
| In this compliant example we utilize a C++ string to buffer the input safely and prevent an overflow. From here we can now safely truncate if we still need it in a c-style string. |
| void function(){  std::string buffer;  std::cin >> buffer; } |

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

| **Principles(s):** ValidateInput Data. This principle maps to this standard because the incoming variable must be checked or risk receiving out of bounds data. The user can not be relied on to provide less than 20 characters. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | Medium to High(Could potentially need logic rework in addition to refactor.) | P9 | L2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | **stream-input-char-array** | Partially checked + soundly supported |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 9.1p0 | **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 |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2024.2 | **CERT\_CPP-STR50-b** **CERT\_CPP-STR50-c** **CERT\_CPP-STR50-e** **CERT\_CPP-STR50-f** **CERT\_CPP-STR50-g** | Avoid overflow due to reading a not zero terminated string Avoid overflow when writing to a buffer Prevent buffer overflows from tainted data Avoid buffer write overflow from tainted data Do not use the 'char' buffer to store input from 'std::cin' |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Prevent SQL injection through input sanitization** |
| --- | --- | --- |
| **SQL Injection** | [STD-004-CPP] | SQL injections can occur when untrusted input is directly included into the database query, allowing attackers to manipulate the SQL statement and bypass authentication or accessed unauthorized data. The safest way of dealing with this is to use parameterized queries, which separate data from code to ensure user input cannot alter the intended structure of the query. |

| **Noncompliant Code** |
| --- |
| In this non-compliant example, we see that the username and the password are being directly combined into the query with no sanitization. Sandwiched together and thrown into the SQL query “as is”, it gives an open door to attackers to use an attack similar to “OR 1=1” to force the logic into a true state. |
| // previous code …  String sqlString = "select \* from db\_user where username=" +          username + " and password =" + pwd;        PreparedStatement stmt = connection.prepareStatement(sqlString);          ResultSet rs = stmt.executeQuery();  // following code … |

| **Compliant Code** |
| --- |
| In this compliant example we see that a prepared statement is used, and the user inputs are bound to the question marks in the statement. This means that no matter what the user typed in, their input would just appear as a string in place of “?”, so the introduction of extra logic would be read as just letters in the username and not commands. |
| // previous code …  String sqlString =        "select \* from db\_user where username=? and password=?";      PreparedStatement stmt = connection.prepareStatement(sqlString);      stmt.setString(1, username);      stmt.setString(2, pwd);      ResultSet rs = stmt.executeQuery();  // following code … |

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

| **Principles(s):** ValidateInput Data. This principle maps to this standard because the incoming query may be an attempt to cause trouble by a bad actor. Parameterizing the query prevents this from happening by treating the user input as a string no matter what it has in it. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | High.(requires logic rework) | P18 | L1 |

**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) |
| [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) | 2025.2 | **SV.DATA.DB**  **SV.SQL** **SV.SQL.DBSOURCE** | Implemented |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Do not access freed memory.** |
| --- | --- | --- |
| **Memory Protection** | [STD-005-CPP] | Accessing memory after it has already been free creates a dangling pointer. This creates undefined behavior and can lead to crashes or the possibility of exploitable vulnerabilities. Once memory is deallocated, all pointers to it become invalid so we must never read or write to it at that point. |

| **Noncompliant Code** |
| --- |
| In this example we have a structure called Example and create a new pointer to a new Example. Then we delete, but after deletion we call a function in Example. |
| Example ex = new Example;  delete ex;  ex->someFunction(); |

| **Compliant Code** |
| --- |
| In this compliant code we are utilizing the stack, instead of the heap to store the new Example. That way when we exit the scope memory is automatically deallocated. |
| Example ex;  ex.someFunction(); |

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

| **Principles(s):** Adopt a Secure Coding Standard. This principle maps to this coding standard because it is taking secure coding into account as the code is developed. By preventing small accidents like this standard we can save time course correcting later. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | Medium (Unlikely to be able to move on with out fixing it, but could lead to logic fixes later) | P9 | L2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Clang) | 3.9 | clang-analyzer-cplusplus.NewDelete clang-analyzer-alpha.security.ArrayBoundV2 | Checked by clang-tidy, but does not catch all violations of this rule. |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 9.1p0 | **ALLOC.UAF** | Use after free |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | **483 S, 484 S** | Partially implemented |
| [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 |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Do not use assertions to check for user or environmental error checking** |
| --- | --- | --- |
| **Assertions** | [STD-006-CPP] | Assertions in C++ should be used to check internal assumptions that should never be violated if the program is correct, and are primarily used for debugging. Since they are disabled in the release they must there for not be relied on for validations for the program to run in the final build. Proper error handling should be used for the build and assertions saved for development. |

| **Noncompliant Code** |
| --- |
| In this non-compliant example, a function takes in two input and divides returns the division of them. The programmer thought to make sure it did not divide by zero using an assert() however that will not show up in the release build so it may allow for divide-by-zero errors given certain input. |
| int divide(int a, int b) {  assert(b != 0);  return a / b;  } |

| **Compliant Code** |
| --- |
| In this compliant example, an error will be thrown if b is 0 preventing the system from entering undefined behavior. |
| int divide(int a, int b) {  if (b == 0) {  throw std::invalid\_argument("divide by zero");  }  return a / b;  } |

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

| **Principles(s):** ValidateInput Data. This principle maps to this standard because the incoming query may be an case of division by 0 which by computer and human hands is both impossible if assertions are used for checking input data. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Medium | Probable | Medium (could require logic fix) | P8 | L2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152428) | 24.04 | **bad-function bad-macro-use** | Supported |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/c/LDRA) | 9.7.1 | **44 S** | Enhanced enforcement |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/c/Parasoft) | 2024.2 | CERT\_C-ERR06-a | Do not use assertions |
| [PC-lint Plus](https://wiki.sei.cmu.edu/confluence/display/c/PC-lint+Plus) | 1.4 | **586** | Fully supported |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Handle all Exceptions.** |
| --- | --- | --- |
| **Exceptions** | [STD-007-CPP] | All exceptions in C++ need to be caught by a matching handler. If not, the program will terminate and abort in order to ensure overall system safety. Even if recovery isn’t possible, it is still best practice to catch these exceptions because it will ensure a controlled termination. |

| **Noncompliant Code** |
| --- |
| In this noncompliant code, a throwingFunction() through an exception and is not handled, thus the program would be forced to terminate. |
| void function() {    throwing\_function();  }    int main() {    function();  } |

| **Compliant Code** |
| --- |
| In this Compliant code, the function is surrounded by a try/catch. This causes the exception that otherwise would have terminated, to be unwound gracefully back to main and complete the rest of the code. |
| void function() {    throwing\_function();  }    int main() {  try {  function();  } catch( … ) {  // Handling the error  }  } |

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

| **Principles(s):** Adopt a Secure Coding Standard. This principle maps to this coding standard because it is taking secure coding into account as the code is developed. By catching the error and not letting it terminate our program for us we can take care of any memory book keeping we may need to by exiting controlled and safe. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Probable | Medium (could require logic rework) | P6 | L2 |

**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 |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 9.1p0 | **LANG.STRUCT.UCTCH PARSE.MBDH** | Masked by handler Masked by default handler |
| [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 |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Range Check Element Access** |
| --- | --- | --- |
| **String Correctness** | [STD-008-CPP] | [Accessing a string element with the operator[] or calling front() or back() to an empty string is unsafe because these function do not perform bound checking, and can therefore lead to undefined behavior if out of range. Always ensure indices are with in stated bounds, and use the at() function when possible since it throws an exception with invalid access naturally. |

| **Noncompliant Code** |
| --- |
| In this noncpompliant code, the value given by int indexNum may be greater than the number of elements stored in the string, resulting in undefined behaviour. |
| void function(int index){  std::string s(“Example”);  s[index] = ‘X’;  } |

| **Compliant Code** |
| --- |
| In this compliant code, we are utilizing the .at() function to throw an error if out of range and then catching it on the next line if there is an issue. |
| void function(int index){  std::string s(“Example”);  try {  s.at(index) = ‘X’;  } catch (std::out\_of\_range &){  //Handle the error  }  } |

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

| **Principles(s):** ValidateInput Data. We should not assume that string s has anything in it at all, let alone at a point “index”. There for we must validate it before we manipulate it. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Unlikely | Medium (could require logic rewrite) | 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 |
| [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) | R2025b | [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 9

| **Coding Standard** | **Label** | **Do need read uninitialized memory.** |
| --- | --- | --- |
| **Data Value** | [STD-009-CPP] | IN C++, reading from uninitialized memory leads to undetermined values, and undefined behavior. Local automatic variables and dynamically allocated memory are not initialized by default so they must be explicitly initialized before use. To avoid errors, always be sure to initialize variables before their values are read. |

| **Noncompliant Code** |
| --- |
| In this noncompliant example, an integer is created, but the value is not initialized, leading to unpredictable output. |
| int example;  std::cout << example; |

| **Compliant Code** |
| --- |
| In this compliant example, when ‘example’ is created, it is immediately initialized to be 0. This prevents unknown values being read and unpredictable behavior. |
| int example = 0;  std::cout << example; |

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

| **Principles(s):** Sanitize Data Sent to Other Systems. Before sending ‘example’ to an outside function like std::cout we much make sure the data is safe to send by initializing it instead of leaving it uninialized. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Probably | High (could be hard to find and require logic rewrite, also high probable) | P12 | L1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | **uninitialized-read** | Partially checked |
| [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 |
| [RuleChecker](https://wiki.sei.cmu.edu/confluence/display/cplusplus/RuleChecker) | 22.10 | **uninitialized-read** | Partially checked |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Close Files when they are no longer needed** |
| --- | --- | --- |
| **File Input**  **/ Output** | [STD-010-CPP] | Every time that std::basic\_filebuf<T>::open() is called it must be paired to a close() either before the object’s lifetime ends, or the program terminates. Doing so ensures resources are released properly. While the higher level classes like ifstream, ofstream, and fstream are safer, and manage their own filebuf open and close, relying on this and not properly closing files can lead to unpredictability. |

| **Noncompliant Code** |
| --- |
| In this noncompliant code, a fstream named someFile has been opened, and things are done to it, but the program is terminated before the stream is closed. |
| std::fstream someFile(someFileName);    if (!someFile.is\_open()) {      // Handle the error      return;    }    // do some things to the file    std::terminate(); |

| **Compliant Code** |
| --- |
| In this compliant example the file stream is closed, and checked to ensure that the close took. This ensures the program will exit properly and resources will be released. |
| std::fstream someFile(someFileName);    if (!someFileName.is\_open()) {      // Handle the error      return;    }    // do some things to the file  someFileName.close();    if (someFileName.fail()) {      // Handle error    }    std::terminate(); |

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

| **Principles(s):** Adopt a Secure Coding Standard. This principle maps to this coding standard because it is taking secure coding into account as the code is developed. Much like when leaving a car we need to close and lock all the doors, we need to also close the stream after we use it, either by closing the scope of utilizing .close(). |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Medium | Unlikely | Low (Quick Fix) | P2 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 9.1p0 | **ALLOC.LEAK** | Leak |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2024.2 | **CERT\_CPP-FIO51-a** | Ensure resources are freed |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2025b | [CERT C++: FIO51-CPP](https://www.mathworks.com/help/bugfinder/ref/certcfio51cpp.html) | Checks for resource leak (rule partially covered) |
| [Security Reviewer - Static Reviewer](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Security+Reviewer+-+Static+Reviewer) | 6.02 | **C80** | Fully implemented |

### 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.

The location in the process that I believe Green Pace should include automated testing is during the build process. If it is done before building, there will not be anything to test. If it is done after the build but before the test is complete, then it will take time to fix things that automatic testing could have caught. Doing the testing even later after the Verify and Test stage will just waste time not catching potential issues that we will then have to verify and test again after changing. On top of timing, it is also part of the philosophy of “Not leaving Security to the end” by including it at several points through out the build process. Also while manual code testing may be time consuming, automated tests allow for quickly checking large swaths of code at a time, meaning it will not appreciably effect build time.

### 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 | Medium | Unlikely | Medium | P4 | L3 |
| STD-003-CPP | High | Likely | Medium to High | P9 | L2 |
| STD-004-CPP | High | Likely | High | P18 | L1 |
| STD-005-CPP | High | Likely | Medium | P9 | L2 |
| STD-006-CPP | Medium | Probable | Medium | P8 | L2 |
| STD-007-CPP | Low | Probable | Medium | P6 | L2 |
| STD-008-CPP | High | Unlikely | Medium | P3 | L3 |
| STD-009-CPP | High | Probably | High | P12 | L1 |
| STD-010-CPP | Medium | Unlikely | Low | P2 | 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 | Policy: All sensitive data must be encrypted with industry standard algorithms like AES-256.  Encryption at rest secures data while it is being stored in databases, disks, or backup. Encrypting at rest data means that if the data becomes compromised, it will be difficult to open. |
| Encryption in flight | Policy: All sensitive data must be sent via a TLS/SSL connection, no unencrypted http or FTP is allowed.  Encryption in flight protects data while it is being transferred. This is a dangerous time for data because of the potential for interception or monitoring via man in the middle attack. This should apply any time data is transferred, both out on the internet, and internally. |
| Encryption in use | Policy: For sensitive data, being actively processed, sensitive data must be worked on in a Trusted Execution Environment.  Encryption in use means that the data needs to be safe while it Is being actively processed. This helps prevent attack from people who have somehow gained system level access. It helps protect data in memory and process from other open processes. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Policy: All users and devices must connect with secure identification mechanisms. This includes multi-factor authentication, strong passwords requirements, and rotation requirements.  Authentication confirms people and devices are who they say they are, this is a basic level of security. |
| Authorization | Policy: Access to systems must follow the principle of least privilege and default deny model. Role based access control will define what a person can access, and frequent reviews will keep the roles updated.  Authorization determines what a user can and can’t do. It applies after authentication and is laid out by the role of the user. |
| Accounting | Policy: All access, config changes, and administrative actions must be logged and timestamped. Logs must be protected from tampering and reviewed regularly.  Accounting is just like auditing. It makes sure people do what they are supposed to, and aren’t doing what they aren’t. In general (from personal experience) they sit in a black box and do nothing most of the time, but are essential if an investigation must take place. |

**\***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 |  |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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