**Green Pace Developer: Security Policy Guide**



# Green Pace Secure Development Policy

## Contents

[Overview 2](#_Toc52464053)

[Purpose 2](#_Toc52464054)

[Scope 2](#_Toc52464055)

[Module Three Milestone 2](#_Toc52464056)

[Ten Core Security Principles 2](#_Toc52464057)

[C/C++ Ten Coding Standards 3](#_Toc52464058)

[Coding Standard 1 4](#_Toc52464059)

[Coding Standard 2 6](#_Toc52464060)

[Coding Standard 3 8](#_Toc52464061)

[Coding Standard 4 10](#_Toc52464062)

[Coding Standard 5 14](#_Toc52464063)

[Coding Standard 6 16](#_Toc52464064)

[Coding Standard 7 18](#_Toc52464065)

[Coding Standard 8 20](#_Toc52464066)

[Coding Standard 9 22](#_Toc52464067)

[Coding Standard 10 24](#_Toc52464068)

[Defense-in-Depth Illustration 26](#_Toc52464069)

[Project One 26](#_Toc52464070)

[1. Revise the C/C++ Standards 26](#_Toc52464071)

[2. Risk Assessment 26](#_Toc52464072)

[3. Automated Detection 26](#_Toc52464073)

[4. Automation 27](#_Toc52464074)

[5. Summary of Risk Assessments 28](#_Toc52464075)

[6. Create Policies for Encryption and Triple A 28](#_Toc52464076)

[7. Map the Principles 30](#_Toc52464077)

[Audit Controls and Management 31](#_Toc52464078)

[Enforcement 31](#_Toc52464079)

[Exceptions Process 31](#_Toc52464080)

[Distribution 32](#_Toc52464081)

[Policy Change Control 32](#_Toc52464082)

[Policy Version History 32](#_Toc52464083)

[Appendix A Lookups 32](#_Toc52464084)

[Approved C/C++ Language Acronyms](#_Toc52464085) 32

Risk Assessment Values 32

## 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 | Treat all data from non-trusted sources as a potential threat until it is validated. Test the type, length, and appropriateness of any data coming into the system to ensure only safe and appropriate data is accepted as valid input while unsafe, inappropriate, or malicious data is not accepted. Ensuring only valid input can interact with the information system. Handle all exceptions that may occur from invalid input to avoid unexpected behavior from the system. |
| 1. Heed Compiler Warnings | Compiler warnings alert developers to potential errors or issues in code that need to be addressed to prevent potential security risks. Ignored warnings could result in software that contains exploitable flaws or unreliable code that generates unexpected results. Consider modifying code that prompts warnings from the compiler to avoid introducing vulnerabilities. |
| 1. Architect and Design for Security Policies | Security is a core requirement and needs to be a primary consideration when planning and designing software architecture. Strive to build security into the foundations rather than attempt to add security layers as an afterthought to ensure integrity of the system. Practice modularization and abstraction with system components to easily accommodate a range of privileges or security authorization levels. |
| 1. Keep It Simple | Increased complexity results in increased risk and a greater number of potential weaknesses vulnerable to exploitation. Systems that are simple are easier to maintain, update, and adapt to protect against threats. Aim to keep software design simple to minimize errors in coding as well as reduce potential errors in use. |
| 1. Default Deny | Protect system assets by defaulting to a denying access unless the access is explicitly granted. This aids in blocking attempts to gain entry or complete requests made from unauthorized parties/users. |
| 1. Adhere to the Principle of Least Privilege | Grant users the lowest access level with fewest permissions needed to accomplish their job functions. Elevated privileges should be granted for only the amount of time required and should default back to minimum after they are no longer needed. This principle limits the harm malicious actors can cause in the event they are able to access a user’s account. |
| 1. Sanitize Data Sent to Other Systems | Ensure outgoing data is transmitted safely, conforms to requirements of the receiving system, is complete, is valid, and is free from harmful elements. This ensure the transmitted data does not cause damage to the receiving system. |
| 1. Practice Defense in Depth | Create multiple security layers to prevent or limit misuse of the system. No single security layer is impenetrable, multiple layers of security are needed to ensure that when one fails another can stop the attack. Adhering to this principle is one of the best ways to protect the organizations systems and data. |
| 1. Use Effective Quality Assurance Techniques | QA ensures that software and IT systems are correctly implemented, tested, and monitored continuously, identifying and addressing vulnerabilities as they arise. Utilize effective QA measure such as JUnit tests, fuzz testing, penetration testing, and security reviews to identify weaknesses or flaws so they can be promptly corrected. |
| 1. Adopt a Secure Coding Standard | Implement standards that developers must comply with when writing code to govern common areas of vulnerability such as input validation, error handling, authentication, and data storage. This will improve consistency and reduce risk of vulnerabilities. |

### 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.

\*\*\*Note: All included standards are from SEI CERT C++ Coding Standards obtained from https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046682 \*\*\*

#### Coding Standard 1

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Type** | STR-001-CPP | Guarantee that storage for strings has sufficient space for character data and the null terminator. |

| **Noncompliant Code** |
| --- |
| This code is noncompliant as the character array for the string storage is limited to 15 characters but the input from the user’s keyboard is unbound. This could lead to buffer overflow if the input is greater than 15 characters. |
| #include <iostream>    **void** function() {  **char** charArray[15];    std::cin >> charArray;  } |

| **Compliant Code** |
| --- |
| To ensure input data is not truncated while guarding against buffer overflows use std::string instead of a bound character array. |
| #include <iostream>  #include <string>    **void** function() {    std::string inputString;    std::cin >> inputString;  } |

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

| **Principles(s):** This standard maps to the principle #1 “Validate Input Data,” by ensuring the storage allocated will be large enough to accommodate the input string. It also maps to #2 “Heed Compiler Warnings,” as most compilers should warn that the code has a potential vulnerability and can lead to buffer overflow. Finally, this standard maps to “Adopt a Secure Coding Standard” (Principle 10) as using std::string is better practice and would be included in a secure coding standard. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Very High (4) | Likely (3) | High (1) | High (12) | 3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion Bauhaus Suite | 7.2.0 | CertC-STR31 | This tool proactively detects potential buffer overflow vulnerabilities stemming from calls to unsafe string functions. It also alerts to potential buffer overruns, including those related to insecure use of fscanf(). |
| Coverity | 2017.07 | STRING\_OVERFLOW  BUFFER\_SIZE  OVERRUN  STRING\_SIZE | Checks that storage size for string is sufficient to store character data and a null terminator. |
| Parasoft C/C++ | 2024.2 | CERT\_C-STR31-a  CERT\_C-STR31-b  CERT\_C-STR31-c  CERT\_C-STR31-d  CERT\_C-STR31-e | Avoids accessing elements of the arrays that are out of bounds. Detects/Prevents overflows when writing to a buffer or untrusted data utilizes buffer. Prevents the use of unsafe string functions that can cause buffer overflow. |
| Security Reviewer – Static Reviewer | 6.02 | RTOS\_33  RTOS\_34  shadowVariable  UNSAFE\_03  UNSAFE\_04 | Guarantees that the storage for strings has sufficient space for character data plus the null terminator. |

Source for this standard: (Mariani, 2025a).

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | INT-001-CPP | Ensure that operations on signed integers do not result in overflow. |

| **Noncompliant Code** |
| --- |
| This noncompliant code for the addition of two signed integers can result in a signed integer overflow. |
| **signed int** function(**signed** **int** int\_a, **signed** **int** int\_b) {  **signed** **int** sum = int\_a + int\_b;  **return** sum;  } |

| **Compliant Code** |
| --- |
| This compliant code checks to ensure limits will not be exceeded prior to executing the addition operation. |
| #include <limits.h>    **signed int** function(**signed** **int** int\_a, **signed** **int** int\_b) {  **signed** **int** sum;  **if** (((int\_b > 0) && (int\_a > (INT\_MAX - int\_b))) ||        ((int\_b < 0) && (int\_a < (INT\_MIN - int\_b)))) {      /\* Handle error \*/    } **else** {      sum = int\_a + int\_b;  **return** sum;    }  } |

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

| **Principles(s):** This standard maps to the principle # 1, ”Validate Input Data” as it requires validating the input variables to ensure performing the operation in question would not cause either an underflow or an overflow. It also applies to this standard “Adopt a Secure Coding Standard” (Principle 10) as guarding against known methods of overflow or underflow is best practice and would be included in a secure coding standard. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High (3) | Likely (3) | Low (3) | Highest (27) | 1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 24.04 | integer-overflow | Ensures that unsigned integer operations do not wrap, that operations on signed integers do not result in overflow and that all integer values are in range. |
| Cppcheck premium | 24.11.0 | premium-cert-int32-c | Ensures that operations on signed integers do not result in overflow. |
| Klocwork | 2025.2 | NUM.OVERFLOW  CWARN.NOEFFECT.OUTOFRANGE  NUM.OVERFLOW.DF | Ensures that unsigned and signed integer operations do not wrap or result in overflow. |
| Parasoft C/C++test | 2024.2 | CERT\_C-INT32-a  CERT\_C-INT32-b  CERT\_C-INT32-c | Avoid signed integer overflows  Integer overflow or underflow in constant expression in '+', '-', '\*' operator  Integer overflow or underflow in constant expression in '<<' operator |

#### Source for this standard: (Seacord, 2025).

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | STR-002-CPP | Use valid references, pointers, and iterators to reference elements of a basic\_string. |

| **Noncompliant Code** |
| --- |
| This code is noncompliant because the iterator loc is invalidated after the first call to insert(), subsequent calls to insert() result in undefined behavior. |
| #include <string>    **void** function(**const** std::string &input) {    std::string email;      // Copy input into email converting ";" to " "    std::string::iterator loc = email.begin();  **for** (auto i = input.begin(), e = input.end(); i != e; ++i, ++loc) {      email.insert(loc, \*i != ';' ? \*i : ' ');    }  } |

| **Compliant Code** |
| --- |
| In this compliant code the value of iterator loc is updated with each call to insert() so an invalid iterator is never accessed. |
| #include <string>    **void** f(**const** std::string &input) {    std::string email;      // Copy input into email converting ";" to " "    std::string::iterator loc = email.begin();  **for** (auto i = input.begin(), e = input.end(); i != e; ++i, ++loc) {      loc = email.insert(loc, \*i != ';' ? \*i : ' ');    }  } |

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

| **Principles(s):** This standard falls beneath the principle “Adopt a Secure Coding Standard” (Principle 10) because use of an invalid pointer is poor practice and can cause unexpected behavior. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High (3) | Probable (2) | High (1) | Medium (6) | 4 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 9.1p0 | ALLOC.UAF | Ensures the use of valid references, pointers, and iterators to reference elements of a basic\_string. |
| Helix QAC | 2025.2 | DF4746, DF4747, DF4748, DF4749 | Ensures the use of valid references, pointers, and iterators to reference elements of a basic\_string. |
| Parasoft C/C++test | 2024.2 | CERT\_CPP-STR52-a | Ensures the use of valid references, pointers, and iterators to reference elements of a basic\_string. |
| Security Reviewer - Static Reviewer | 6.02 | C24 | Ensures the use of valid references, pointers, and iterators to reference elements of a basic\_string. |

Source for this standard: (Mariani, 2025b).

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | IDS-001-J | Prevent SQL Injection |

| **Noncompliant Code** |
| --- |
| This noncompliant code uses java.sql.PrepareStatement incorrectly and still allows a SQL injection attack by incorporating the unsanitized input arguments username and password into the prepared statement. |
| **import** java.sql.Connection;  **import** java.sql.DriverManager;  **import** java.sql.ResultSet;  **import** java.sql.SQLException;  **import** java.sql.Statement;    **class** Login {  **public** Connection getConnection() **throws** SQLException {      DriverManager.registerDriver(**new**              com.microsoft.sqlserver.jdbc.SQLServerDriver());      String dbConnection =        PropertyManager.getProperty("db.connection");      // Can hold some value like      // "jdbc:microsoft:sqlserver://<HOST>:1433,<UID>,<PWD>"  **return** DriverManager.getConnection(dbConnection);    }    **public** **void** doPrivilegedAction(  **String** username, **String** password) **throws** SQLException {      Connection connection = getConnection();  **if** (connection == **null**) {        // Handle error      }  **try** {        String sqlString = "select \* from db\_user where username=" +          username + " and password =" + password;        PreparedStatement stmt = connection.prepareStatement(sqlString);          ResultSet rs = stmt.executeQuery();  **if** (!rs.next()) {  **throw** **new** SecurityException("User name or password incorrect");        }          // Authenticated; proceed      } **finally** {  **try** {          connection.close();        } **catch** (SQLException x) {          // Forward to handler        }      }    }  } |

| **Compliant Code** |
| --- |
| This compliant code parameterized the username and password to correctly use the java.sql.PrepareStatement to safely pass arguments and prevent SQL injection. |
| **import** java.sql.Connection;  **import** java.sql.DriverManager;  **import** java.sql.ResultSet;  **import** java.sql.SQLException;  **import** java.sql.Statement;    **class** Login {  **public** Connection getConnection() **throws** SQLException {      DriverManager.registerDriver(**new**              com.microsoft.sqlserver.jdbc.SQLServerDriver());      String dbConnection =        PropertyManager.getProperty("db.connection");      // Can hold some value like      // "jdbc:microsoft:sqlserver://<HOST>:1433,<UID>,<PWD>"  **return** DriverManager.getConnection(dbConnection);    }    **public** **void** doPrivilegedAction(  **String** username, **String** password) **throws** SQLException {      Connection connection = getConnection();  **if** (connection == **null**) {        // Handle error      }  **try** {        String sqlString = "select \* from db\_user where username=? And password=?";        PreparedStatement stmt = connection.prepareStatement(sqlString);   stmt.setString(1, username);   stmt.setString(2, password);        ResultSet rs = stmt.executeQuery();  **if** (!rs.next()) {  **throw** **new** SecurityException("User name or password incorrect");        }          // Authenticated; proceed      } **finally** {  **try** {          connection.close();        } **catch** (SQLException x) {          // Forward to handler        }      }    }  } |

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

| **Principles(s):** This standard maps to many principles, the first being “Validate Input Data” (principle 1) as the query input should be validated to ensure it is safe (does not include “ OR “ for example). It maps to “Architect and Design for Security Policies” (principle 3) as the design should parameterize the SQL query to prevent SQL injection. It also maps to “Sanitize Data Sent to Other Systems” (principle 7) as the code sends the data in the form of a query to the other system, the SQL database, and should ensure it is safe to do so. Finally, it maps to the standard “Adopt a Secure Coding Standard” (Principle 10) as guarding against known SQL injection is best practice and would be included in a secure coding standard. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High (3) | Likely (2) | Low (3) | Very High (18) | 2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 9.0p0 | JAVA.IO.INJ.SQL | Guards against SQL injection. |
| Coverity | 7.5 | SQLI  FB.SQL\_PREPARED\_STATEMENT\_GENERATED\_  FB.SQL\_NONCONSTANT\_STRING\_PASSED\_TO\_EXECUTE | Prevents SQL injection. |
| Klocwork | 2025.2 | SV.DATA.DB  SV.SQL  SV.SQL.DBSOURCE | Prevents SQL injection. |
| SpotBugs | 4.6.0 | SQL\_NONCONSTANT\_STRING\_PASSED\_TO\_EXECUTE  SQL\_PREPARED\_STATEMENT\_GENERATED\_FROM\_NONCONSTANT\_STRING | Prevents SQL injection. |

#### Source for this standard: (Mohindra, 2025).

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | MEM-001-CPP | Detect and handle memory allocation errors. |

| **Noncompliant Code** |
| --- |
| If allocation fails in this noncompliant code it could lead to unexpected termination of the program. |
| #include <cstring>    **void** function(**const** **int** \*array, std::**size\_t** size) noexcept {  **int** \*copy = **new** **int**[size];    std::**memcpy**(copy, array, size \* **sizeof**(\*copy));    // ...  **delete** [] copy;  } |

| **Compliant Code** |
| --- |
| This compliant code checks for a nullptr and handles it if one is returned. |
| #include <cstring>  #include <new>    **void** f(**const** **int** \*array, std::**size\_t** size) noexcept {  **int** \*copy = **new** (std::**nothrow**) **int**[size];  **if** (!copy) {      // Handle error  **return**;    }    std::**memcpy**(copy, array, size \* **sizeof**(\*copy));    // ...  **delete** [] copy;  } |

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

| **Principles(s):** This standard maps to principle 1, “Validate Input Data” as it is validating the pointer input into a function is valid, an invalid pointer as input into a function that expects a valid pointer can lead to unexpected program termination. It also maps to “Architect and Design for Security Policies” (principle 3) as we are building the check into the design from the beginning to prevent issues. Finally this standard maps to “Adopt a Secure Coding Standard” (#10) as the input validation and proper error handling for memory allocation errors are part of secure coding standards. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High (3) | Likely | Low | Highest (27) | 1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Coverity | 7.5 | CHECKED\_RETURN | Finds inconsistencies in how function call return values are handled. |
| KlocWork | 2025.2 | NPD.CHECK.CALL.MIGHT  NPD.CHECK.CALL.MUST  NPD.CHECK.MIGHT  NPD.CHECK.MUST  NPD.CONST.CALL  NPD.CONST.DEREF  NPD.FUNC.CALL.MIGHT  NPD.FUNC.CALL.MUST  NPD.FUNC.MIGHT  NPD.FUNC.MUST  NPD.GEN.CALL.MIGHT  NPD.GEN.CALL.MUST  NPD.GEN.MIGHT  NPD.GEN.MUST  RNPD.CALL  RNPD.DEREF | Detect and handle memory allocation errors. |
| Parasoft C/C++ test | 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 ++ |  |  | Runtime detection. |

Source for this standard: (Pincar, 2025c).

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | MSC-001-J | Do not use assertions to verify the absence of runtime errors. |

| **Noncompliant Code** |
| --- |
| Do not user assertions in systems in deployment, malicious users can trigger a failed assertion in a denial of service attack. This noncompliant code example uses an assert statement to verify input was received. |
| **BufferedReader** buffer\_reader;    /\*set up buffer reader \*/    **String** line = buffer\_reader.readLine();  **assert** line != **null**; |

| **Compliant Code** |
| --- |
| Compliant code uses an if statement instead to detect and handle unavailable input. |
| **BufferedReader** buffer\_reader;    /\*set up buffer reader \*/    **String** line = buffer\_reader.readLine();  **if (**line == **null**) {  // handle error  } |

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

| **Principles(s):** This standard maps to principle 10, “Adopt a Secure Coding Standard” as asserts are removed from production builds and as such good practice, included in secure coding standards, is to avoid using assert statements to verify lack of runtime errors. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low (1) | Probable (2) | Medium (2) | Low (4) | 5 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Parasoft Jtest | 2024.2 | CERT.MSC60.ASSERT | Do not use assertions in production code. |
| “In general, the misuse of the assert statement for runtime checking rather than checking for logical errors cannot be detected automatically.” (Long, 2023). | | | |

#### Source for this standard: (Long, 2023)

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | ERR-001-CPP | Handle all exceptions. |

| **Noncompliant Code** |
| --- |
| In the noncompliant code below the exception thrown is not caught and results in an unexpected termination of the program. |
| void throws\_exception() noexcept(false);    void do\_something() {  throws\_exception();  }    int main() {  do\_something();  } |

| **Compliant Code** |
| --- |
| In this example code the exception is caught and handled in the do\_something function. An alternative solution is the try catch block can be handled in the main function. |
| void throws\_exception() noexcept(false);    void do\_something() {  try {  throws\_exception();  } catch (…) {  //Handle error  }  }    int main() {  do\_something();  } |

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

| **Principles(s):** This standard maps to principle 10, “Adopt a Secure Coding Standard” as detecting and handling exceptions is good practice and is included in a secure coding standard. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low (1) | Probable (2) | Low (3) | Medium (6) | 4 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion Bauhaus Suite | 7.2.0 | CertC++-ERR51 | Ensures code handles all exceptions. |
| Klocwork | 2025.2 | MISRA.CATCH.ALL | Ensures code handles all exceptions. |
| Parasoft C/C++ test | 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. |
| Security Reviewer – Static Reviewer | 6.02 | C35 | Fully implemented to ensure code handles all exceptions. |

#### Source for this standard: (Ballman, 2025a)

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Input Output** | FIO-001-CPP | Close files when they are no longer needed. |

| **Noncompliant Code** |
| --- |
| In the noncompliant code below the file is not appropriately closed as std::terminate() uses std::abort() which does not call destructors. |
| #include <exception>  #include <fstream>  #include <string>    **void** function(**const** std::string &fileName) {    std::fstream file(fileName);  **if** (!file.is\_open()) {      // Handle error  **return**;    }    // ...    std::terminate();  } |

| **Compliant Code** |
| --- |
| In the compliant example below the file is properly closed before std::terminate() is called. |
| #include <exception>  #include <fstream>  #include <string>    **void** function(**const** std::string &fileName) {    std::fstream file(fileName);  **if** (!file.is\_open()) {      // Handle error  **return**;    }    // ...  file.close();  **if** (file.fail()) {      // Handle error    }    std::terminate();  } |

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

| **Principles(s):** This standard maps to principle 10, “Adopt a Secure Coding Standard” as closing files when they are no longer needed prevents memory leaks and is best practice included in a secure coding standard. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Medium (2) | Unlikely (1) | High (1) | Low (2) | 5 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 9.1p0 | ALLOC.LEAK | Checks that files are closed when they are no longer needed. |
| KlocWork | 2025.2 | RH.LEAK | Checks that files are closed when they are no longer needed. |
| Parasoft C/C++ test | 2024.2 | CERT\_CPP-FIO51-a | Ensure resources are freed |
| Security Reviewer – Static Reviewer | 6.02 | C80 | Fully implemented - Checks that files are closed when they are no longer needed. |

Source for this standard: (Pincar, 2025b).

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Expressions** | EXP-001-CPP | Do not read uninitialized memory. |

| **Noncompliant Code** |
| --- |
| The integer declared\_int below is declared but not initialized before being included in the cout expression. This noncompliant code results in undefined behavior. |
| #include <iostream>    **void** function() {  **int** declared\_int;    std::cout << declared\_int;  } |

| **Compliant Code** |
| --- |
| In the code below the declared\_int is declared and initialized before being included in the print statement. |
| #include <iostream>    **void** function() {  **int** declared\_int = 0;    std::cout << declared\_int;  } |

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

| **Principles(s):** This standard falls beneath the principle “Adopt a Secure Coding Standard” (Principle 10) because use of a variable before initialization is poor practice that leads to undefined behavior. Avoiding this error is best practice and included in a secure coding standard. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High (3) | Probable (2) | Medium (2) | High (12) | 3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 9.1p0 | LANG.STRUCT.RPL  LANG.MEM.UVAR | Return pointer to local Uninitialized variable |
| Klocwork | 2025.2 | UNINIT.CTOR.MIGHT UNINIT.CTOR.MUST UNINIT.HEAP.MIGHT UNINIT.HEAP.MUST UNINIT.STACK.ARRAY.MIGHT UNINIT.STACK.ARRAY.MUST UNINIT.STACK.ARRAY.PARTIAL.MUST UNINIT.STACK.MIGHT UNINIT.STACK.MUST | Ensures that the code does not read uninitialized memory. |
| Parasoft C/C++ test | 2024.2 | CERT\_CPP-EXP53-a | Ensures that the code avoids use before initialization. |
| Security Reviewer – Static Reviewer | 6.02 | C54 C55 C56 C57 C58 C59 C60 C61 C62 C63 | Fully implemented - Ensures that the code does not read uninitialized memory. |

Source for this standard: (Pincar, 2025a).

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Functions** | MSC-002-CPP | Value-returning functions must return a value from all exit paths. |

| **Noncompliant Code** |
| --- |
| In the noncompliant code below if the integer is greater than or equal to zero the function does not return a required boolean value resulting in undefined behavior. |
| **bool** is\_Negative(**int** integer) {  **if** (integer < 0) {  **return** true;    }  } |

| **Compliant Code** |
| --- |
| In the compliant code example below both exit paths return appropriate boolean values. |
| **bool** is\_Negative(**int** integer) {  **if** (integer < 0) {  **return** true;    }  **return** false;  } |

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

| **Principles(s):** This standard maps to principle 3 “Architect and Design for Security Policies” and principle 10, “Adopt a Secure Coding Standard” as failing to return a value from a code path that expects a return value, results in undefined behavior that can be exploited to corrupt data integrity. Careful design ensures that a return value is provided for all code paths and is part of a secure coding standard. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Medium (2) | Probable (2) | Medium (2) | Medium (8) | 4 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 9.1p0 | LANG.STRUCT.MRS  LANG.STRUCT.NVNR | Checks for missing return statement.  Checks for non-void no return. |
| Klocwork | 2025.2 | FUNCRET.GEN  FUNCRET.IMPLICIT | Checks that value returning functions return a value from all exit paths. |
| Parasoft C/C++ test | 2024.2 | CERT\_CPP-MISC52-a | Ensures all exit paths from a function, except main(), with non-void return type shall have an explicit return statement with an expression. |
| Security Reviewer – Static Reviewer | 6.02 | RTOS\_09  RTOS\_10  RTOS\_11  RTOS\_12 | Fully implemented – ensures that value returning functions return a value from all exit paths. |

Source for this standard: (Ballman, 2025b)

### 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 focus on security needs to occur in all stages of development as illustrated in the image above. In the assess and plan stage developers need to analyze the threat landscape as well as the regulations that may apply to the project. When updates are needed to respond to new threats developers need to assess and plan carefully how they can correct software flaws and mitigate the threats. In the design and build phase developers need to plan for security by following best practices which include utilizing test-driven designs. Designing and building for security includes building security into the system’s foundations by practicing modularization and abstraction with system components to easily accommodate a range of privileges or security authorization levels. It also includes strict adherence to the secure coding standard adopted by Green Pace.

In is vital to add automated testing including static analysis to the DevOps process to move to DevSecOps this can be done by adding it in the verify and testing phase of software development. The verify and testing phase should also include unit testing, testing of the application as a whole, and having the system analyzed by security experts to uncover additional vulnerabilities prior to release. As illustrated in the image above security remains a priority and is a major component of the post release phases. In the transition and health check phase security settings are verified and penetration testing is completed. During the monitor and detect phase the system’s many layers of defenses are monitored for abnormalities so that any intrusion is detected as quickly as possible. In the event of a threat steps are taken during the respond phase to block the attack and roll back the system if needed to return to safe operations. During maintain and stabilize the system is assessed against its established security baseline and steps are taken to return to baseline following an attack.

### 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 |
| --- | --- | --- | --- | --- | --- |
| MEM-001-CPP | High | Likely | Low | Highest | 1 |
| INT-001-CPP | High | Likely | Low | Highest | 1 |
| IDS-001-J | High | Likely | Low | Very High | 2 |
| STD-001-CPP | Very High | Likely | High | High | 3 |
| EXP-001-CPP | High | Probable | Medium | High | 3 |
| STR-002-CPP | High | Probable | High | Medium | 4 |
| MSC-002-CPP | Medium | Probable | Medium | Medium | 4 |
| ERR-001-CPP | Low | Probable | Low | Medium | 4 |
| FIO-001-CPP | Medium | Unlikely | High | Low | 5 |
| MSC-001-J | Low | Probable | Medium | Low | 5 |

### 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 | Data is considered at rest when it is being stored (for example on a database) and not actively being utilized. Encryption at rest is a policy that states data at rest needs to be encrypted. This policy adds an additional layer of data in the event a malicious attacker gains access to the stored data. Adherence to encryption at rest requires the attacker would either need access to the encryption key or would have to spend a large amount of time using brute force to decrypt the data. This layer of protection makes it harder for attackers to steal sensitive information. Green Pace will utilize AES-256 to encrypt all data at rest and the security team will be responsible for creating and managing encryption keys. |
| Encryption in flight | Data is considered in flight when it is in transit, either moving through the system, for example the account balance being pulled from the database to be displayed on the application, or moving outside of the system for example being sent to a partner application. Data in flight is vulnerable to interception or modification while in transit. Encryption in flight is a policy that states data in transit will be encrypted making it less vulnerable to attack. It includes utilizing protocols like transport layer security (TLS), virtual private networks (VPNs) and SSH file transport protocol (SFTP). Green Pace will utilize encryption in flight for all data in transit. The security team will be responsible for creating and managing keys. |
| Encryption in use | Data is considered in use when it is actively being utilized by the system. Encryption in use is a policy that states data in use will be encrypted to add an additional layer of security. Protecting data in use involves use of tools such as homomorphic encryption or secure multi-party computation to perform calculation on encrypted data bypassing the need to decrypt it. Encryption in use will be used by Green Pace for all sensitive data like passwords or account numbers. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication is the security process that attempts to ensure the user is who they claim to be. Green Pace will use multifactor authentication for all users who are attempting to access the system. The first factor for authentication will be either be logging in with matching username and password or matching biometrics (fingerprint, facial recognition) if enabled on the device. The second factor for authentication will be a one-time use code that will be sent via text message to the user on their registered number. The passwords will be required to be more than 14 characters with at least 1 uppercase, 1 lowercase, 1 symbol and 1 number. One time use codes will be 6 digits long and only good for 5 minutes after the code has been requested. If the credentials fail to match the user can attempt again. If the user fails to match more than 3 times the user will receive a notification via email and/or text message and the account will be locked until the user completes a secure account unlock process. Once authenticated the username can be logged to track all actions initiated by the user. User login credentials will always be protected by adherence to the encryption policies. |
| Authorization | Authorization is the process of determining what activities and resources an authenticated user is allowed to utilize. At Green Pace the users will be authorized for the minimum privileges necessary to accomplish each users’ tasks. If elevated permissions are needed temporarily the request will be approved by administration and the security team prior to being granted. The special permission will only remain in effect as long as required before being dropped to the user’s normal authorization level. Green Pace will have a default deny mentality that will block a user from functionality unless explicitly granted by the user’s authorization level. Adherence to this policy minimizes the damage a bad actor can cause if they gain access to the system via stolen credentials. The addition of new users and changes to users’ level of access will be logged and monitored for excessive changes that may indicate the system has been infiltrated by bad actors. Suspected compromised accounts will have permissions removed until safety has been ensured by the security team. |
| Accounting | Accounting refers to logging and tracking the services and resources each user has accessed or attempted to access. Logs will be monitored for unusual activity. Particularly logs will include user login attempts, addition of new users, changes to users’ level of access, which files are accessed by the user, and any changes made to the database. The logs will be included the time of request, if the request was successful and the duration of the activity of session. Successful accounting is essential to a secure system, in the event of an attack this allows the security team to analyze how the breech occurred, what the attackers gained access to, how to stop the attacker, and how to mitigate future risk via the method used by the attacker. |

**\***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.

\*\*\*Principle mapping is included in the space provided in the coding standards on pages 3-25.

**NOTE:** Green Pace has already successfully implemented the following:

* Operating system logs
* Firewall logs
* Anti-malware logsThe 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 | 07/20/2025 | Completed 10 principles and started to add coding standards | Connie Knupp |  |
| 1.2 | 08/10/2025 | Completed missing items in security policy. | Connie Knupp |  |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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

## Risk Assessment Values

**Severity** – how serious the consequences are if the rule is not followed.

|  |  |  |
| --- | --- | --- |
| **Determination** | **Value** | **Example Consequences** |
| **Low** | 1 | Denial of service attacks, abnormal program termination |
| **Medium** | 2 | Data integrity violation, unintentional information disclosure |
| **High** | 3 | Run arbitrary code |
| **Very High** | 4 |

**Likelihood** – how probable an exploitation results from the flaw if rule is violated.

|  |  |
| --- | --- |
| **Determination** | **Value** |
| **Unlikely** | 1 |
| **Probable** | 2 |
| **Likely** | 3 |

**Remediation Cost** – numeric value determined by ease of detection and ability to repair without impacting subsequent code functionality.

|  |  |  |
| --- | --- | --- |
| **Determination** | **Value** | **Meaning** |
| **Low** | 3 | Easy to detect and easy to repair without harming subsequent code function. |
| **Medium** | 2 | Either easy to detect **or** easy to repair without harming subsequent code function (both are not true). |
| **High** | 1 | Difficult to detect and difficult to repair without inspecting subsequent code for impact on subsequent code functionality. |

**Priorities and Levels** – how quickly this item should be detected and corrected. This value is determined by multiplying the severity, the likelihood, and the remediation costs.

|  |  |  |
| --- | --- | --- |
| **Priority Label** | **Priority Value (severity \* likelihood \* cost).** | **Level** |
| **Low** | Less than or equal to 5. | 5 |
| **Medium** | Greater than or equal to 6 and less than 12. | 4 |
| **High** | Greater than or equal to 12 and less than 18. | 3 |
| **Very High** | Greater than or equal to 18 and less than 24. | 2 |
| **Highest** | Greater than or equal to 24. | 1 |

**References**

Ballman, A. (2025a, July 21). *Err51-CPP. handle all exceptions*. Carnegie Mellon University: Software Engineering Institute. <https://wiki.sei.cmu.edu/confluence/display/cplusplus/ERR51-CPP.+Handle+all+exceptions>

Ballman, A. (2025b, July 8). *MSC52-CPP. value-returning functions must return a value from all exit paths*. Carnegie Mellon University: Software Engineering Institute. . <https://wiki.sei.cmu.edu/confluence/display/cplusplus/MSC52-CPP.+Value-returning+functions+must+return+a+value+from+all+exit+paths>

Long, F. (2023, January 10). *MSC60-J. Do not use assertions to verify the absence of runtime errors.* Carnegie Mellon University: Software Engineering Institute. <https://wiki.sei.cmu.edu/confluence/display/java/MSC60-J.+Do+not+use+assertions+to+verify+the+absence+of+runtime+errors>

Mariani, F. (2025a, August 6). *Str31-C. Guarantee that storage for strings has sufficient space for character data and the null terminator*. Carnegie Mellon University: Software Engineering Institute. <https://wiki.sei.cmu.edu/confluence/display/c/STR31-C.+Guarantee+that+storage+for+strings+has+sufficient+space+for+character+data+and+the+null+terminator>

Mariani, F. (2025b, July 2). *Str52-CPP. Use valid references, pointers, and iterators to reference elements of a basic\_string*. Carnegie Mellon University: Software Engineering Institute. <https://wiki.sei.cmu.edu/confluence/display/cplusplus/STR52-CPP.+Use+valid+references%2C+pointers%2C+and+iterators+to+reference+elements+of+a+basic_string>

Mohindra, D. (2025, August 6). *IDS00-J. Prevent SQL Injection*. Carnegie Mellon University: Software Engineering Institute. <https://wiki.sei.cmu.edu/confluence/display/java/IDS00-J.+Prevent+SQL+injection>

Pincar, J. (2025a, July 25). *Exp53-CPP. do not read uninitialized memory*. Carnegie Mellon University: Software Engineering Institute. <https://wiki.sei.cmu.edu/confluence/display/cplusplus/EXP53-CPP.+Do+not+read+uninitialized+memory>

Pincar, J. (2025b, July 2). *Fio51-CPP. close files when they are no longer needed*. Carnegie Mellon University: Software Engineering Institute. <https://wiki.sei.cmu.edu/confluence/display/cplusplus/FIO51-CPP.+Close+files+when+they+are+no+longer+needed>

Pincar, J. (2025c, June 25). *Mem52-CPP. detect and handle memory allocation errors*. Carnegie Mellon University: Software Engineering Institute. <https://wiki.sei.cmu.edu/confluence/display/cplusplus/MEM52-CPP.+Detect+and+handle+memory+allocation+errors>

Seacord, R. (2025, May 5). *INT32-C. ensure that operations on signed integers do not result in overflow*. Carnegie Mellon University: Software Engineering Institute. <https://wiki.sei.cmu.edu/confluence/display/c/INT32-C.+Ensure+that+operations+on+signed+integers+do+not+result+in+overflow>