**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 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 | Cross stie scripting, SQL injection, and command injection are common security threats. Input validation verifies the values provided are what programmer expect before access is granted. |
| 1. Heed Compiler Warnings | The warnings within the IDE’s compiler will give developers warning and messages about what code may cause errors. |
| 1. Architect and Design for Security Policies | Architecture is the strategic design of systems, technology, and policies. These are used to protect businesses and IT companies from cyber threats. |
| 1. Keep It Simple | Applying security practices and principals should be effortless, and not require too much consideration. |
| 1. Default Deny | Application requests, network traffic, and system access will all be denied until a specifically approved predefined rule has been set in place. |
| 1. Adhere to the Principle of Least Privilege | Users and entities have access for only what is required to complete tasks. This includes access rights to specified data. |
| 1. Sanitize Data Sent to Other Systems | The sensitive data and information will be cleared from the system storage. According to the (DSL) data security lifecycle, this is very important. |
| 1. Practice Defense in Depth | Defense in depth leverages multiple layers of security. This helps protect an organization’s assets. |
| 1. Use Effective Quality Assurance | It is important to check for quality of the product/ Otherwise there may be technical failures or missed requirements within the product. |
| 1. Adopt a Secure Coding Standard | Have a set of rules in place that are predefined to protect assets and information. Without security standards, it will be hard to stay of track with protective measures. |

### 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** | **Implement abstract data types using opaque types** |
| --- | --- | --- |
| **Data Type** | [STD-001-CPP] | Abstract data types work more efficiently when used with a private or (opaque) data type. |

| **Noncompliant Code** |
| --- |
| This code is based on the managed string library developed by CERT [Burch 2006]. The managed string type and the functions that operate on this type are defined in the string\_m.h header file. |
| **Struct** string\_mx {  **size\_t** size;  **size\_t** maxsize;    unsigned **char** strtype;  **char** \*cstr;  };    **typedef** **struct** string\_mx string\_mx;    /\* Function declarations \*/  **extern** errno\_t strcpy\_m(string\_mx \*s1, **const** string\_mx \*s2);  **extern** errno\_t strcat\_m(string\_mx \*s1, **const** string\_mx \*s2);  /\* … \*/ |

| **Compliant Code** |
| --- |
| The solution reimplements the string\_mx type, hiding the implementation data type from the user of the managed string library. To get this done, the developer of a private data types creates two header files: an external string\_m.h header file that is included by the user of the data type and an internal file that is included only in files that implement the managed string abstract data type, |
| |  | | --- | | **struct** string\_mx;  **typedef** **struct** string\_mx string\_mx;    /\* Function declarations \*/  **extern** errno\_t strcpy\_m(string\_mx \*s1, **const** string\_mx \*s2);  **extern** errno\_t strcat\_m(string\_mx \*s1, **const** string\_mx \*s2);  /\* … \*/ |   In the internal header file, struct string\_mx is fully defined but not visible to a user of the data abstraction:   |  | | --- | | **struct** string\_mx {  **size\_t** size;  **size\_t** maxsize;    unsigned **char** strtype;  **char** \*cstr;  }; | |

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

| **Principles(s):** **Implement abstract data types using opaque types – data types work better when used with a private data type** |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Unlikely | High | P1 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion Bauhaus suite | 7.2.0 | CertC- DCL12 | None |
| LDRA tool suite | 9.7.1 | 104 D | Partially implemented |
| Polyspace Bug Finder | R2024a | Cert C: Rec. DCL12-C | Checks for object structure in the file of the pointer to the object is not dereferenced |
| Parasoft C/C++test | 2023.1 | Cert\_CDCL12-a | If a pointer to an object is not dereferenced, the object implementation must be hidden |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Do not encode noncharacter data as a string** |
| --- | --- | --- |
| **Data Value** | [STD-002-CPP] | Usually, programmers will store arbitrary data into strings. The data within the strings will not be valid. |

| **Noncompliant Code** |
| --- |
| The code attempts to convert a big integer into a string value. Then, restore into the big integer value. Restoring after conversion will ikely output a different value than that the original. |
| BigInteger x = **new** BigInteger(“530500452766”);  **byte**[] byteArray = x.toByteArray();  String s = **new** String(byteArray);  byteArray = s.getBytes();  x = **new** BigInteger(byteArray); |

| **Compliant Code** |
| --- |
| BigInteger x = **new** BigInteger(“530500452766”);  String s = x.toString();  // Valid character data  **byte**[] byteArray = s.getBytes();  String ns = **new** String(byteArray);  x = **new** BigInteger(ns);  BigInteger x = **new** BigInteger(“530500452766”);  **byte**[] byteArray = x.toByteArray();  String s = Base64.getEncoder().encodeToString(byteArray);  byteArray = Base64.getDecoder().decode(s);  x = **new** BigInteger(byteArray); |
| [Compliant code block; code should be indented using 12-point Courier New font.] |

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

| **Principles(s):** Do not encode noncharacter data as a string – If non characters are implemented in a string, the code will be invalid. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| None | None | None | None |
| None | None | None | None |
| None | None | None | None |
| None | None | None | None |

#### 

#### Coding Standard 3

| **Coding Standard** | **Label** | **Understand how escape characters are interpreted when string are loaded** |
| --- | --- | --- |
| **String Correctness** | [STD-003-CPP] | This drives home how important it is to ensure the user sees what is needed in the correct format. |

| **Noncompliant Code** |
| --- |
| The code defines a method that finds matches between the string literals and the input. |
| **public** **class** Splitter {    // Interpreted as backspace    // Fails to split on word boundaries  **private** **final** String WORDS = "\b";    **public** String[] splitWords(String input) {      Pattern pattern = Pattern.compile(WORDS);      String[] input\_array = pattern.split(input);  **return** input\_array;    }  } |

| **Compliant Code** |
| --- |
| The code below correctly displays the escaped value of a string literal. |
| **public** **class** Splitter {    // Interpreted as two chars, '\' and 'b'    // Correctly splits on word boundaries  **private** **final** String WORDS = "\\b";    **public** String[] split(String input){      Pattern pattern = Pattern.compile(WORDS);      String[] input\_array = pattern.split(input);  **return** input\_array;    }  } |

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

| **Principles(s):** using escape characters correctly allows the users to see what was intended for them to see. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| None | None | None | None | None |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| The checker Framework | 2.1.3 | Tainting Checker | Trust and security errors |
| None | None | None | None |
| None | None | None | None |
| None | None | None | None |

#### 

#### Coding Standard 4

| **Coding Standard** | **Label** | **Prevent SQL Injection** |
| --- | --- | --- |
| **SQL Injection** | [STD-004-JAVA] | Maliciously altered queries are the culprit of security risks for all companies. The goal of SQL injection is to validate and sanitize procedures. |

| **Noncompliant Code** |
| --- |
| The code permits a SQL Injection attack by using an unsensitized input argument in the SQL command. The attacker can inject SQL code. |
| **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);    }      String hashPassword(**char**[] password) {      // Create hash of password    }    **public** **void** doPrivilegedAction(String username, **char**[] password)  **throws** SQLException {      Connection connection = getConnection();  **if** (connection == **null**) {        // Handle error      }  **try** {        String pwd = hashPassword(password);          String sqlString = "SELECT \* FROM db\_user WHERE username = '"                           + username +                           "' AND password = '" + pwd + "'";        Statement stmt = connection.createStatement();        ResultSet rs = stmt.executeQuery(sqlString);    **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** |
| --- |
| With the use of perimetric queries, a question mark can be used as a placeholder for the argument. The code validates the size of the username to prevent the attacker from submitting a name that is too long. |
| **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);    }      String hashPassword(**char**[] password) {      // Create hash of password    }    **public** **void** doPrivilegedAction(String username, **char**[] password)  **throws** SQLException {      Connection connection = getConnection();  **if** (connection == **null**) {        // Handle error      }  **try** {        String pwd = hashPassword(password);          String sqlString = "SELECT \* FROM db\_user WHERE username = '"                           + username +                           "' AND password = '" + pwd + "'";        Statement stmt = connection.createStatement();        ResultSet rs = stmt.executeQuery(sqlString);    **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):** Prevent SQL injection by sanitizing the procedures |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| The Checker Framework | 2.1.3 | Tainting Checker | Trust and security errors |
| CodeSonar | 8.1p0 | JAVA.IO.INJ.SQL | SQL Injection |
| Coverity | 7.5 | **SQLI FB.SQL\_PREPARED\_STATEMENT\_GENERATED\_** **FB.SQL\_NONCONSTANT\_STRING\_PASSED\_TO\_EXECUTE** | Implemented |
| FindBugs | 1.0 | **SQL\_NONCONSTANT\_STRING\_PASSED\_TO\_EXECUTE** | Implemented |
| Fortify | 1.0 | **HTTP\_Response\_Splitting** **SQL\_Injection\_\_Persistence** **SQL\_Injection** | Implemented |
| KlocWork | 2024.2 | **SV.DATA.DB** **SV.SQL** **SV.SQL.DBSOURCE** | SQL Injection |
| Parasoft Jtest | 2024.1 | CERT.IDSOO.TDSQL | Protect against SQL Injection |
| SonarQube | 9.9 | S2077  S3649 | Executing queries is security-sensitive  SQL queries should not be vulnerable to injection attacks |

#### 

#### Coding Standard 5

| **Coding Standard** | **Label** | **Do not read uninitialized memory** |
| --- | --- | --- |
| **Memory Protection** | [STD-005-CPP] | Automatic variables assume unexpected values if they are read before initialization. |

| **Noncompliant Code** |
| --- |
| The code below shows the programmer forgot to attest for the case number when it equals zero. It is uninitialized. Meaning, the behavior will be undefined. |
| **void** set\_flag(**int** number, **int** \*sign\_flag) {  **if** (NULL == sign\_flag) {  **return**;    }    **if** (number > 0) {      \*sign\_flag = 1;    } **else** **if** (number < 0) {      \*sign\_flag = -1;    }  }    **int** is\_negative(**int** number) {  **int** sign;    set\_flag(number, &sign);  **return** sign < 0;  } |

| **Compliant Code** |
| --- |
| This repairs the issue by accounting for a possible instance a number can qual zero. |
| **void** set\_flag(**int** number, **int** \*sign\_flag) {  **if** (NULL == sign\_flag) {  **return**;    }      /\* Account for number being 0 \*/  **if** (number >= 0) {      \*sign\_flag = 1;    } **else** {      \*sign\_flag = -1;    }  }    **int** is\_negative(**int** number) {  **int** sign = 0; /\* Initialize for defense-in-depth \*/    set\_flag(number, &sign);  **return** sign < 0;  } |

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

| **Principles(s):** Protection of memory will ensure the initialization of data will not equal null. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 24.04 | Uninitialized-local-read  Uninitialized-variable-use | Fully checked |
| Coverity | 1.66 | UNINIT | Implemented |
| LDRA tool suite | 9.7.1 | 53 d, 69 D, 631 S, 652 S | Fully implemented |
| Polyspace Bug finder | R2024a | CERT C: Rule EXP33-C | Non-initialized variable  Non-initialized pointer |

#### 

#### Coding Standard 6

| **Coding Standard** | **Label** | **Use a static assertion to test the value of constant expressions** |
| --- | --- | --- |
| **Assertions** | [STD-006-CPP] | Assertions offer great value for developing applications. They get rid of software defects that lead to vulnerabilities. |

| **Noncompliant Code** |
| --- |
| The code uses assert for a property regarding memory-mapped structuring for proper code behavior. |
| #include <assert.h>    **struct** timer {    unsigned **char** MODE;    unsigned **int** DATA;    unsigned **int** COUNT;  };    **int** func(**void**) {  **assert**(**sizeof**(**struct** timer) == **sizeof**(unsigned **char**) + **sizeof**(unsigned **int**) + **sizeof**(unsigned **int**));  } |

| **Compliant Code** |
| --- |
| [Compliant description] |
| **struct** timer {    unsigned **char** MODE;    unsigned **int** DATA;    unsigned **int** COUNT;  };    #if (sizeof(struct timer) != (sizeof(unsigned char) + sizeof(unsigned int) + sizeof(unsigned int)))    #error “Structure must not have any padding”  #endif  #include <assert.h>    **struct** timer {    unsigned **char** MODE;    unsigned **int** DATA;    unsigned **int** COUNT;  };    static\_assert(**sizeof**(**struct** timer) == **sizeof**(unsigned **char**) + **sizeof**(unsigned **int**) + **sizeof**(unsigned **int**),                “Structure must not have any padding”); |

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

| **Principles(s):** Static assertions ensure vulnerability are kept to at bay from software defects |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Unlikely | High | P1 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion Bauhaus suite | 7.2.0 | CertC-DCl03 | None |
| Clang | 3.9 | Misc-static-assert | Checked by clang-tidy |
| CodeSonar | 8.1P0 | Customization | User can enter a custom check to report the uses of the assert() |
| ÉCLAIR | 1.2 | CC2.DCL03 | Fully implemented |

#### 

#### Coding Standard 7

| **Coding Standard** | **Label** | **Honor exception specifications** |
| --- | --- | --- |
| **Exceptions** | [STD-007- CPP] | If a function throws an exception other than what was pre-determined, it may lead to termination of the program. |

| **Noncompliant Code** |
| --- |
| The function declared as not throwing. The memory requested cannot be allocated. |
| #include <cstddef>  #include <vector>    **void** f(std::vector<**int**> &v, **size\_t** s) noexcept(**true**) {    v.resize(s); // May throw  } |

| **Compliant Code** |
| --- |
| The no except spec is removed. This allows for the function to allow all exceptions. |
| #include <cstddef>  #include <vector>    **void** f(std::vector<**int**> &v, **size\_t** s) {    v.resize(s); // May throw, but that is okay  } |

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

| **Principles(s):** Honoring exception speciation will prevent non predetermined exceptions to be thrown |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 22.10 | Un-handled-throw-exception | Partially Checked |
| Helix QAC | 2024.2 | **C++4035, C++4036, C++4632** | None |
| Rulchecker | 22.10 | Un-handled-throw-exception | Partially checked |
| Polyspace Bugfinder | R202a | CERT C++: ERR55-CPP | Checks for no except functions exiting with exception (the rule is fully covered |

#### 

#### Coding Standard 8

| **Coding Standard** | **Label** | **Use valid iterator ranges** |
| --- | --- | --- |
| Loops | [STD-008-CPP] | When developers iterate through containers. The iterators that are in use must iterate over a valid range. If not, could cause an infinite loop. |

| **Noncompliant Code** |
| --- |
| The code below there are two iterators. If the iterators are not equal the loop will continue to iterate. This results in undefined behavior. |
| #include <algorithm>  #include <iostream>  #include <vector>    **void** f(**const** std::vector<**int**> &c) {    std::for\_each(c.end(), c.begin(), [](**int** i) { std::cout << i; });  } |

| **Compliant Code** |
| --- |
| The code below uses the iterator values passed to the std followed by the scope operator. Are in the proper order for the iterators to work effectivly. |
| #include <algorithm>  #include <iostream>  #include <vector>    **void** f(**const** std::vector<**int**> &c) {    std::for\_each(c.begin(), c.end(), [](**int** i) { std::cout << i; });  } |

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

| **Principles(s):** valid iterator ranges will ensure programs will not break due to an infinite loop |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 22.10 | Overflow-upon-dereference | None |
| Codesonar | 8.1P0 | LANG.MEM.BO | Buffer overturn |
| Parasoft C/C++test | 2023.1 | **CERT\_CPP-CTR53-a** **CERT\_CPP-CTR53-b** | Do not use and iterator that is not in a range.  Do not compare iterators from different containers |
| PVS-Studio | 7.32 | V539, V662, V789 | None |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Do not access an object outside its lifetime** |
| --- | --- | --- |
| Object access | [STD-009-CPP] | If this is done, it results in undefined behavior. |

| **Noncompliant Code** |
| --- |
| Using a pointer to point to an object, we can call the non-static function of the object. Before the beginning of the pointer’s lifetime. |
| **struct** S {  **void** mem\_fn();  };    **void** f() {    S \*s;    s->mem\_fn();  } |

| **Compliant Code** |
| --- |
| In the code below storage is obtained. This is for the pointer before calling the S scope operator mem\_fn() |
| **struct** S {  **void** mem\_fn();  };    **void** f() {    S \*s = **new** S;    s->mem\_fn();  **delete** s;  } |

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

| **Principles(s):** Accessing objects outside their lifetimes can cause program to render some elements undefined |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 22.10 | **return-reference-local dangling\_pointer\_use** | Partially checked |
| Clang | 3.9 | **-Wdangling-initializer-list** | Catches some lifetime issues related to incorrect use of the std::initializer list |
| Codesonar | 8.1P0 | IO.UAC  ALLOC.UAF | User after close  Use after free |
| Heliz QAC | 2024.2 | **C++4003, C++4026**  **DF2812, DF2813, DF2814, DF2930, DF2931, DF2932, DF2933, DF2934,** | None |

Coding Standard 10

| **Coding Standard** | **Label** | **Do not use a bitwise operator with a Boolean operand** |
| --- | --- | --- |
| Bitwise usage | [STD-010-LLL] | Intertwingling the relational operands is a sign of logical error. This comes from the programmer. |

| **Noncompliant Code** |
| --- |
| Bitwise and operator is used with the results of two expressions. |
| **if** (getuid() == 0 & getgid() == 0) {    /\* ... \*/  } |

| **Compliant Code** |
| --- |
| The code below uses both “&&” for the logical operation with the conditional expression. |
| **if** (getuid() == 0 && getgid() == 0) {    /\* ... \*/  } |

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

| **Principles(s):** Using a bitwise value with a Boolean operand causes programmer logical issues |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 24.04 | Inappropriate-bool | Supported indirectly via MISRA C:2012 Rule 10.1 |
| Codesonar | 8.1P0 | LANG.TYPE.IOT | Inappropriate operand type |
| Klocwork | 2024.2 | **MISRA.LOGIC.OPERATOR.NOT\_BOOL** | None |
| Polyspace bug finder | R202a4 | Cert C: Rule EXP46-C | None |

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 security policies outlined above all play a role in protecting the integrity and assets of companies. The principals are to help employees maintain best security practices such that it will not be an after though when developing code. Without the security principal embedded into the workflow will cause serious issues. Testing measures are important, and it is imperative to continually test throughout the DevSecOps lifecycle. This protects companies from malicious hackers and software defects BEFORE the deployment stage. If companies have security policies in place, the chance of companies losing assets and other sensitive information will dramatically be decreased.

### 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 | High | P1 | L3 |
| STD-002-CPP | Low | Unlikely | Medium | P2 | L3 |
| STD-003-CPP | None | None | None | None | None |
| STD-004-CPP | High | Likely | Medium | P18 | L1 |
| STD-005-CPP | High | Probable | Medium | P12 | L1 |
| STD-006-CPP | Low | Unlikely | High | P1 | L3 |
| STD-007-CPP | Low | Likely | Low | P9 | L2 |
| STD-008-CPP | High | Probable | High | P6 | L2 |
| STD-009-CPP | High | Probable | High | P6 | L2 |
| STD-010-CPP | Low | Likely | Low | P9 | L2 |

### Create Policies for Encryption and Triple A

Include all three types of encryptions (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 | Used to convert text into cipher text. Without the decryption key, the information will be unreadable. Authorized users must use the decryption key. This is used to protect sensitive data from unauthorized users or interception of data onto a disk or removeable drive. s |
| Encryption in flight | During content transfer, protocol uses feedback from the cypher along with checksum to verify integrity of data in transit or in flight. This is used for when data needs to be transferred without compromising its security. The reason is to maintain data integrity in mid-flight to another location. Such as a disk, or removeable drive. |
| Encryption in use | This ensures sensitive data is always secured across all types of encryptions during each step of the DevSecOps lifecycle. This gives more opportunities for sharing, using, and monetization of data. This is used throughout the whole process in DevSecOps to protect encrypted data when in use. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | This verifies a person is who they say they are. The reason this applies is to ensure sensitive data is not compromised. This may also include the addition of new users. |
| Authorization | This serves to give the correct persons the authority to access specific documentation, coupled with sensitive information. This applies because we need to give the correct person access to only what they need to complete their work. This includes logins, passwords, database changes, and the level of user access. |
| Accounting | The documentation of who accessed data at what time using what commands. This applies because it is important to maintain a history of what was done. If there is an audit, user can access the records to view the performed actions. This includes the files accessed by the user. |

### 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 always follow this policy.

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 | 08/10/2020 | Initial Template | Kendall White | Green Pace Mngmnt |
| [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 |