**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 | Ensure that data input is validated for type, length, format, and range before processing. This can help prevent injection attacks and prevent software vulnerabilities. Validating data input will ensure the data is what the application expects. |
| 1. Heed Compiler Warnings | Utilize the highest warning level available for your compiler. Make sure to address all warnings as they are identifying potential vulnerabilities that could lead to security breaches. |
| 1. Architect and Design for Security Policies | Incorporate security policies from the beginning stages of design. A proactive approach to security will ensure that it is an integral part of architect and design. |
| 1. Keep It Simple | Keep the software design as simple as possible. Utilizing complex designs can increase the possibility of errors and make it more difficult to secure and maintain. |
| 1. Default Deny | Denial of access will be default and only explicitly granted as necessary. This protection scheme will ensure that proper conditions are met before granting access permission. |
| 1. Adhere to the Principle of Least Privilege | Only the minimum level of access need to complete their functions will be granted. If higher access is needed, it will be granted for the least amount of time needed to complete the privileged task and only the minimum access needed will be granted. |
| 1. Sanitize Data Sent to Other Systems | Ensure that any data passed through other systems are sanitized in order to prevent the ability of injection attacks or data leaks. This includes removing or encoding harmful elements. |
| 1. Practice Defense in Depth | Employ multiple layers of security strategies so if one layer fails or is inadequate, another layer of security will be in place to mitigate the risk. Having multiple layers of security will reduce the likelihood of successful exploit. |
| 1. Use Effective Quality Assurance Techniques | Implement thorough testing and quality assurance early in the development process. Identifying and eliminating vulnerabilities will be completed utilizing code reviews, static analysis, and dynamic testing. |
| 1. Model threats | Utilize threat modeling to anticipate possible threat that the software my be subjected to. Utilizing threat modeling identifies key assets, decomposing the application, identifying and categorizing the threats to each asset. |

### 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** | **Name of Standard** |
| --- | --- | --- |
| **Data Type** | [STD-001-CPP] | Do not cast to an out-of-range enumeration value:  Adhering to INT50-CPP standard helps avoid undefined behavior caused by casting to out-of-range enumeration values. |

| **Noncompliant Code** |
| --- |
| The integer is cast to an enumeration type without ensuring the integer is within the valid range of the enumeration before cast. |
| enum EnumType {    First,    Second,    Third  };    void f(int intVar) {    EnumType enumVar = static\_cast<EnumType>(intVar);      if (enumVar < First || enumVar > Third) {      // Handle error    }  } |

| **Compliant Code** |
| --- |
| The integer is checked to ensure that the integer is within the valid range of the enumeration before it is cast. |
| enum EnumType {    First,    Second,    Third  };    void f(int intVar) {    if (intVar < First || intVar > Third) {      // Handle error    }    EnumType enumVar = static\_cast<EnumType>(intVar);  } |

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

| **Principles(s):** INT50-CPP is a type safety principle that prohibits casting an enumeration value that is outside the range of the valid enumerators for that type. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.10 | cast-integer-to-enum |  |
| Parasoft C/C++test | 2023.1 | CERT\_CPP-INT50-a | An expression with enum underlying type shall only have values corresponding to the enumerators of the enumeration |
| Polyspace Bug Finder | R2024 | CERT C++: INT50-CPP | Checks for casting to out-of-range enumeration value (rule fully covered) |
| CodeSonar | 8.1p0 | LANG.CAST.COERCE  LANG.CAST.VALUE | Coercion Alters Value  Cast Alters Value |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | [STD-002-CPP] | Do not declare or define a reserved identifier:  Adhering to DCL51-CPP will ensure the code remains robust, maintainable, and compatible with future updates to the C++ standard library. |

| **Noncompliant Code** |
| --- |
| This Noncompliant code is declaring “” x as a user-defined literal. |
| #include <cstddef>    unsigned int operator"" x(const char \*, std::size\_t); |

| **Compliant Code** |
| --- |
| This Compliant code is declaring “” \_x as a user-defined literal which is not a reserved identifier. |
| #include <cstddef>    unsigned int operator"" \_x(const char \*, std::size\_t); |

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

| **Principles(s):** DCL51-CPP is in place to avoid name conflicts and preventing undefined behavior. Adhering to this principle will help ensure that code will remain compliant with future versions of C++. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Helix QAC | 2024.2 | C++5003 |  |
| LDRA tool suite | 9.7.1 | 86 S, 218 S, 219 S, 580 S | Fully implemented |
| Parasoft C/C++test | 2023.1 | CERT\_CPP-DCL51-a  CERT\_CPP-DCL51-b  CERT\_CPP-DCL51-c  CERT\_CPP-DCL51-d  CERT\_CPP-DCL51-e  CERT\_CPP-DCL51-f | Do not #define or #undef identifiers with names which start with underscore Do not redefine reserved words Do not #define nor #undef identifier 'defined' The names of standard library macros, objects and functions shall not be reused The names of standard library macros, objects and functions shall not be reused (C90) The names of standard library macros, objects and functions shall not be reused (C99) |
| Polyspace Bug Finder | R2024a | CERT C++: DCL51-CPP | Checks for redefinitions of reserved identifiers (rule partially covered) |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | [STD-003-CPP] | Do not attempt to create a std::string from a null pointer:  Adhering to STR51-CPP will prevent undefined behavior by ensuring that the constructor never receives a null pointer. |

| **Noncompliant Code** |
| --- |
| In this Noncompliant code example, a ‘std::string’ oject is created directly from the results of the call to ‘std::getenv()’ which risks the return of a null pointer. This can lead to unexpected behavior. |
| #include <cstdlib>  #include <string>    void f() {    std::string tmp(std::getenv("TMP"));    if (!tmp.empty()) {      // ...    }  } |

| **Compliant Code** |
| --- |
| In this compliant solution, the result from the call to ‘std::getenv()’ is checked for null before the std::string object is constructed. This ensures that the ‘std::string’ constructor never receives a null pointer preventing undefined behavior. |
| #include <cstdlib>  #include <string>    void f() {    const char \*tmpPtrVal = std::getenv("TMP");    std::string tmp(tmpPtrVal ? tmpPtrVal : "");    if (!tmp.empty()) {      // ...    }  } |

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

| **Principles(s):** STR51-CPP is used to ensure that std::string from a null pointer. When a std::string is created from a null pointer it can lead to undefined behavior such as crashes. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.10 | assert\_failure |  |
| CodeSonar | 8.1p0 | LANG.MEM.NPD | Null Pointer Dereference |
| Parasoft C/C++test | 2023.1 | CERT\_CPP-STR51-a | Avoid null pointer dereferencing |
| Polyspace Bug Finder | R2024a | CERT C++: STR51-CPP | Checks for string operations on null pointer (rule partially covered). |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | [STD-004-JAV] | Prevent SQL injection:  Adhering to IDS00-J will help prevent SQL injection by properly escaping input values. |

| **Noncompliant Code** |
| --- |
| In this noncompliant code the SQL query string is built linking the user inputs allowing an attacker to manipulate it. |
| 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;        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** |
| --- |
| In this compliant code a parameterized query is utilized to securely handle the user inputs. |
| public void doPrivilegedAction(    String username, char[] password  ) throws SQLException {    Connection connection = getConnection();    if (connection == null) {      // Handle error    }    try {      String pwd = hashPassword(password);        // Validate username length      if (username.length() > 8) {        // Handle error      }        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();      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):** IDS00-J is utilize to ensure that preventing SQL injection by relying on validating and sanitizing user inputs before they are included in SQL queries. |
| --- |

**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 (see Chapter 8) |
| CodeSonar | 8.1p0 | JAVA.IO.INJ.SQL | SQL Injection (Java) |
| Parasoft Jtest | 2024.1 | CERT.IDS00.TDSQL | Protect against SQL injection |
| SonarQube | 9.9 | S2077  S3649 | Executing SQL queries is security-sensitive  SQL queries should not be vulnerable to injection attacks |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | [STD-005-CPP] | Properly deallocate dynamically allocated resources:  Adhering to MEM51-CPP will help avoid potential issues related to uninitialized pointers which could lead to undefined behavior. |

| **Noncompliant Code** |
| --- |
| This noncompliant code does not initialized i1 or i2 which could lead to undefined behavior when trying to delete as one of these could hold an invalid address. |
| #include <new>    void f() {    int \*i1, \*i2;    try {      i1 = new int;      i2 = new int;    } catch (std::bad\_alloc &) {      delete i1;      delete i2;    }  } |

| **Compliant Code** |
| --- |
| In this compliant code i1 and i2 are initialized to nullptr which means if allocations fails, the pointers remain nullptr which is a safe value to pass to delete. |
| #include <new>    void f() {    int \*i1 = nullptr, \*i2 = nullptr;    try {      i1 = new int;      i2 = new int;    } catch (std::bad\_alloc &) {      delete i1;      delete i2;    }  } |

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

| **Principles(s):** MEM51-CPP. Properly deallocate dynamically allocated resources – This rule emphasizes the importance of managing and releasing resources. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Parasoft C/C++test | 2023.1 | CERT\_CPP-MEM51-a  CERT\_CPP-MEM51-b  CERT\_CPP-MEM51-c  CERT\_CPP-MEM51-d | Use the same form in corresponding calls to new/malloc and delete/free  Always provide empty brackets ([]) for delete when deallocating arrays  Both copy constructor and copy assignment operator should be declared for classes with a nontrivial destructor  Properly deallocate dynamically allocated resources |
| Parasoft Insure++ |  |  | Runtime detection |
| Polyspace Bug Finder | R2024a | CERT C++: MEM51-CPP | Checks for:  Invalid deletion of pointer  Invalid free of pointer  Deallocation of previously deallocated pointer  Rule partially covered. |
| PVS-Studio | 7.32 | V515, V554, V611, V701, V748, V773, V1066 |  |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | [STD-006-CLG] | Use a static assertion to test the value of a constant expression:  Adhering to DCL03-C will ensure that structure padding issues are detected at compile time allowing for clear diagnostics and avoid runtime overhead. |

| **Noncompliant Code** |
| --- |
| In this noncompliant code the check is happening at runtime which requires code execution to detect padding. This assertion is placed in a function, not close to the definition making is less clear and harder to maintain. |
| #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** |
| --- |
| In this compliant code the static assertions are evaluated at compile time and provides a clear error message directly in the source code if the condition fails. |
| #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):** DCL03-C is used to ensure the correctness in the code. This is done by verifying constant expressions at compile time instead of runtime. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Clang | 3.9 | misc-static-assert | Checked by clang-tidy |
| CodeSonar | 8.1p0 | (customization) | Users can implement a custom check that reports uses of the assert() macro |
| Compass/ROSE |  |  | Could detect violations of this rule merely by looking for calls to assert(), and if it can evaluate the assertion (due to all values being known at compile time), then the code should use static-assert instead; this assumes ROSE can recognize macro invocation |
| LDRA tool suite | 9.7.1 | 44 S | Fully implemented |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | [STD-007-CPP] | Handle all exceptions thrown before main() begins executing:  Adhering to ERR58-CPP will ensure that exceptions are manage properly and that the program does not terminate unexpectedly due to unhandled exceptions during global object construction. |

| **Noncompliant Code** |
| --- |
| In this noncompliant code the global object ‘globalS’ is constructed during startup and at this point exception handling is not well defined. If an exception is thrown at this time, the program will be terminated. |
| struct S {    S() noexcept(false);  };    static S globalS; |

| **Compliant Code** |
| --- |
| In this compliant code the global object ‘globalS’ is initialized only when the ‘globalS’ function is called for the first time. This allows the proper handling of the error and avoids abrupt termination. |
| struct S {    S() noexcept(false);  };    S &globalS() {    try {      static S s;      return s;    } catch (...) {      // Handle error, perhaps by logging it and gracefully terminating the application.    }    // Unreachable.  } |

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

| **Principles(s):** ERR58-CPP is used to ensure that any exceptions that are thrown prior to the execution of ‘main’ are handled properly so the system does not proceed with execution in an unsafe state. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 8.1p0 | LANG.STRUCT.EXCP.THROW | Use of throw |
| Parasoft C/C++test | 2023.1 | CERT\_CPP-ERR58-a | Exceptions shall be raised only after start-up and before termination of the program |
| Polyspace Bug Finder | R2024a | CERT C++: ERR58-CPP | Checks for exceptions raised during program startup (rule fully covered) |
| RuleChecker | 22.10 | potentially-throwing-static-initialization | Partially checked |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Input Output (FIO) | [STD-008-CPP] | Close files when they are no longer needed:  Adhering to FIO51-CPP will ensure that the file resources are properly released even if the program is terminated abruptly. |

| **Noncompliant Code** |
| --- |
| This noncompliant code could result in unclosed file because it was not properly closed. This can lead to resource leaks and potential corruption of the file. |
| #include <exception>  #include <fstream>  #include <string>    void f(const std::string &fileName) {    std::fstream file(fileName);    if (!file.is\_open()) {      // Handle error      return;    }    // ...    std::terminate();  } |

| **Compliant Code** |
| --- |
| This compliant code ensures that the file is properly closed, which is crucial for resource management and avoiding potential data corruption or leaks. |
| #include <exception>  #include <fstream>  #include <string>    void f(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):** FIO51-CPP enforces the discipline of closing files as soon as they are no longer needed. This will promote better resource management and efficiency. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 8.1p0 | ALLOC.LEAK | Leak |
| Parasoft C/C++test | 2023.1 | CERT\_CPP-FIO51-a | Ensure resources are freed |
| Parasoft Insure++ |  |  | Runtime detection |
| Polyspace Bug Finder | R2024a | CERT C++: FIO51-CPP | Checks for resource leak (rule partially covered) |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Characters and Strings (STR) | [STD-009-CPP] | Guarantee that storage for strings has sufficient space for character data and the null terminator:  Adhering to STR50-CPP ensures that inputs are handled dynamically and avoiding issues with buffer overflow. |

| **Noncompliant Code** |
| --- |
| This noncompliant code reads input from the standard input stream which can only hold 12 characters and if the user inputs more characters the input will result in a buffer overflow leading to potential undefined behavior or vulnerabilities. |
| #include <iostream>    void f() {    char buf[12];    std::cin >> buf;  } |

| **Compliant Code** |
| --- |
| This compliant code handles dynamic memory allocation by automatically resizes to accommodate the input data without risk of overflow. |
| #include <iostream>  #include <string>    void f() {    std::string input;    std::string stringOne, stringTwo;    std::cin >> stringOne >> stringTwo;  } |

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

| **Principles(s):** STR50-CPP ensures that string operations are safe and do not inadvertently overwrite adjacent memory. This mitigates memory related vulnerabilities. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 8.1p0 | MISC.MEM.NTERM  LANG.MEM.BO  LANG.MEM.TO | No space for null terminator  Buffer overrun Type overrun |
| Parasoft C/C++test | 2023.1 | 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' |
| Polyspace Bug Finder | R2024a | CERT C++: STR50-CPP | Checks for:  Use of dangerous standard function  Missing null in string array  Buffer overflow from incorrect string format specifier  Destination buffer overflow in string manipulation  Insufficient destination buffer size  Rule partially covered. |
| SonarQube C/C++ Plugin | 4.10 | S3519 |  |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Miscellaneious (MSC) | [STD-010-CLG] | Never hard code sensitive information:  Adhering to MSC41-C will ensure that security is not vulnerable when handling sensitive data. |

| **Noncompliant Code** |
| --- |
| This noncompliant code directly passes a hardcoded string to authenticate which is a security risk as it can be easily extracted. It also has no user input making it impractical for real world applications. |
| /\* Returns nonzero if authenticated \*/  int authenticate(const char\* code);    int main() {    if (!authenticate("correct code")) {      printf("Authentication error\n");      return -1;    }      printf("Authentication successful\n");    // ...Work with system...    return 0;  } |

| **Compliant Code** |
| --- |
| This compliant code prompts the user to enter their authentication code. After using the authentication conde the buffer is cleared in order to prevent sensitive data from lingering in memory. |
| /\* Returns nonzero if authenticated \*/  int authenticate(const char\* code);    int main() {  #define CODE\_LEN 50    char code[CODE\_LEN];    printf("Please enter your authentication code:\n");    fgets(code, sizeof(code), stdin);    int flag = authenticate(code);    memset\_s(code, sizeof(code), 0, sizeof(code));    if (!flag) {      printf("Access denied\n");      return -1;    }    printf("Access granted\n");    // ...Work with system...    return 0;  } |

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

| **Principles(s):** MSC41-C advises against hard coding sensitive information. This aligns with principle of least privilege, defense in depth, security by design, and confidentiality. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 8.1p0 | HARDCODED.AUTH  HARDCODED.DNS  HARDCODED.KEY  HARDCODED.SALT  HARDCODED.SEED | Hardcoded Authentication Hardcoded DNS Name Hardcoded Crypto Key Hardcoded Crypto Salt Hardcoded Seed in PRNG |
| Parasoft C/C++test | 2023.1 | CERT\_C-MSC41-a | Do not hard code string literals |
| PC-lint Plus | 1.4 | 2460 | Assistance provided: reports when a literal is provided as an argument to a function parameter with the ‘noliteral’ argument Semantic; several Windows API functions are marked as such and the ‘-sem’ option can apply it to other functions as appropriate |
| Polyspace Bug Finder | R2024a | CERT C: Rule MSC41-C | Checks for hard coded sensitive data (rule partially covered) |

### Defense-in-Depth Illustration

This illustration provides a visual representation of the defense-in-depth best practice of layered security.



## Project One

There are seven steps outlined below that align with the elements you will be graded on in the accompanying rubric. When you complete these steps, you will have finished the security policy.

### Revise the C/C++ Standards

You completed one of these tables for each of your standards in the Module Three milestone. In Project One, add revisions to improve the explanation and examples as needed. Add rows to accommodate additional examples of compliant and noncompliant code. Coding standards begin on the security policy.

### Risk Assessment

Complete this section on the coding standards tables. Enter high, medium, or low for each of the headers, then rate it overall using a scale from 1 to 5, 5 being the greatest threat. You will address each of the seven policy standards. Fill in the columns of severity, likelihood, remediation cost, priority, and level using the values provided in the appendix.

### Automated Detection

Complete this section of each table on the coding standards to show the tools that may be used to detect issues. Provide the tool name, version, checker, and description. List one or more tools that can automatically detect this issue and its version number, name of the rule or check (preferably with link), and any relevant comments or description—if any. This table ties to a specific C++ coding standard.

### Automation

Provide a written explanation using the image provided.



Automation will be used for the enforcement of and compliance to the standards defined in this policy. Green Pace already has a well-established DevOps process and infrastructure. Define guidance on where and how to modify the existing DevOps process to automate enforcement of the standards in this policy. Use the DevSecOps diagram and provide an explanation using that diagram as context.

Utilizing Green Pace’s already well-established DevOps process and infrastructure, we can integrate automated enforcement and compliance checks throughout the process. I recommend starting early in the development process to ensure all policies are adhered to throughout the entire process.

Assess and Planning: Utilize automated change impact analysis to evaluate the security implications of planned changes will reduce the risk of unintentional vulnerabilities being created.

Design: Implement automated static analysis tools that enforce secure coding practices are adhered to within the design documents. This will ensure that security considerations are met such as API security, input validation, and error handling.

Build: Implement continuous integration pipelines that automatically check security of third party libraries and scan for vulnerabilities.

Verify and Test: Utilize automated testing tools to perform vulnerability scanning, penetration testing, and security compliance testing.

Respond: Utilize automated scripts to automatically block attacks, shut down compromised services, or roll back to a previous state.

### 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 | Medium | Unlikely | Medium | P4 | L3 |
| STD-002-CPP | Low | Unlikely | Low | P3 | L3 |
| STD-003-CPP | High | Likely | Medium | P18 | L1 |
| STD-004-JAV | High | Likely | Medium | P18 | L1 |
| STD-005-CPP | High | Likely | Medium | P18 | L1 |
| STD-006-CLG | Low | Unlikely | High | P1 | L3 |
| STD-007-CPP | Low | Likely | Low | P9 | L2 |
| STD-008-CPP | Medium | Unlikely | Medium | P4 | L3 |
| STD-009-CPP | High | Likely | Medium | P18 | L1 |
| STD-010-CLG | High | Probable | Medium | P12 | L1 |

### 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 | Encryption at rest is the concept of encrypting data that is being stored on a device such as a hard drive or server. Having this data encrypted will help prevent the information from being taken by unauthorized users in the event of a breach. Having the data encrypted will require an unauthorized user to break the encryption in order to view the data. |
| Encryption in flight | Encryption in flight refers to encrypting data that is being moved over a network. Utilizing encryption in flight will reduce the likelihood of a malicious attacker intercepting and being able to read the data being sent. This is completed by each side transmission having a unique encryption key that allows only the sender and received being able to see the data. |
| Encryption in use | Encryption in use refers to data being encrypted even when being used. To do so, the data is/can be manipulated as if it is decoded data. Even if an attacker were to gain access to this stream of data, they will not be able to do anything with is since it is encrypted. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication is the practice of having a user provide information about who they are. This is then compared to a database that houses usernames, passwords, or any other form of authentication tool such as USB key, fingerprint, of PKI card with password. Authentication needs to be updated on a regular basis with new users coming in, users going out, and change in users level of access. It is recommended to utilize a Multi-Factor Authentication tool to strengthen security. Such MFA would be a two step authentication like a user name and password, but then a code is sent to a designated phone number that will then need to be entered in order to gain access. |
| Authorization | Authorization is the granting of specified privileges to certain areas of a network or system. Once a user is authenticated, they will only be able to access the areas that they were authorized to go to. This access should only be granted with the principle of least privilege. The principle of least privilege is only granting access to the areas that the user needs to complete their task. This privilege can be extended to other areas that they normally would not have access, but should only be granted for as long as they need to complete a specific task. |
| Accounting | Accounting is the act of logging any system changes and resource access. An example of this would be logging what a user reads or writes too and what they are changing. Depending on the frequency that this log is reviewed will determine how effective it would be to discovering a breach. If it is infrequently reviewed, it could take weeks/months to discover the breach, which is why review of logs should be completed frequently and can even be automated. |

**\***Use this checklist for the Triple A to be sure you include these elements in your policy:

* User logins
* Changes to the database
* Addition of new users
* User level of access
* Files accessed by users

### Map the Principles

Map the principles to each of the standards, and provide a justification for the connection between the two. In the Module Three milestone, you added definitions for each of the 10 principles provided. Now it’s time to connect the standards to principles to show how they are supported by principles. You may have more than one principle for each standard, and the principles may be used more than once. Principles are numbered 1 through 10. You will list the number or numbers that apply to each standard, then explain how each of these principles supports the standard. This exercise demonstrates that you have based your security policy on widely accepted principles. Linking principles to standards is a best practice.

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

* Operating system logs
* Firewall logs
* Anti-malware logs

The only item you must complete beyond this point is the Policy Version History table.

## Audit Controls and Management

Every software development effort must be able to provide evidence of compliance for each software deployed into any Green Pace managed environment.

Evidence will include the following:

* Code compliance to standards
* Well-documented access-control strategies, with sampled evidence of compliance
* Well-documented data-control standards defining the expected security posture of data at rest, in flight, and in use
* Historical evidence of sustained practice (emails, logs, audits, meeting notes)

## Enforcement

The office of the chief information security officer (OCISO) will enforce awareness and compliance of this policy, producing reports for the risk management committee (RMC) to review monthly. Every system deployed in any environment operated by Green Pace is expected to be in compliance with this policy at all times.

Staff members, consultants, or employees found in violation of this policy will be subject to disciplinary action, up to and including termination.

## Exceptions Process

Any exception to the standards in this policy must be requested in writing with the following information:

* Business or technical rationale
* Risk impact analysis
* Risk mitigation analysis
* Plan to come into compliance
* Date for when the plan to come into compliance will be completed

Approval for any exception must be granted by chief information officer (CIO) and the chief information security officer (CISO) or their appointed delegates of officer level.

Exceptions will remain on file with the office of the CISO, which will administer and govern compliance.

## Distribution

This policy is to be distributed to all Green Pace IT staff annually. All IT staff will need to certify acceptance and awareness of this policy annually.

## Policy Change Control

This policy will be automatically reviewed annually, no later than 365 days from the last revision date. Further, it will be reviewed in response to regulatory or compliance changes, and on demand as determined by the OCISO.

## Policy Version History

| Version | Date | Description | Edited By | Approved By |
| --- | --- | --- | --- | --- |
| 1.0 | 08/05/2020 | Initial Template | David Buksbaum |  |
| 1.1 | 07/21/2024 | Module 3 changes | Chris King |  |
| 1.2 | 08/11/2024 | Project 1 Changes | Chris King |  |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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