**Green Pace Developer: Security Policy Guide Template**



# Green Pace Secure Development Policy

## Contents

Overview 2

Purpose 2

Scope 2

Module Three Milestone 2

Ten Core Security Principles 2

C/C++ Ten Coding Standards 3

Coding Standard 1 4

Coding Standard 2 5

Coding Standard 3 6

Coding Standard 4 7

Coding Standard 5 8

Coding Standard 6 9

Coding Standard 7 10

Coding Standard 8 11

Coding Standard 9 13

Coding Standard 10 14

Defense-in-Depth Illustration 15

Project One 15

1. Revise the C/C++ Standards 15

2. Risk Assessment 15

3. Automated Detection 15

4. Automation 15

5. Summary of Risk Assessments 16

6. Create Policies for Encryption and Triple A 16

7. Map the Principles 17

Audit Controls and Management 18

Enforcement 18

Exceptions Process 18

Distribution 19

Policy Change Control 19

Policy Version History 19

Appendix A Lookups 19

Approved C/C++ Language Acronyms 19

## 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.

## 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 | This principle validates any input from external sources meets any requirements. This is one of the most important steps because if the information entered is not formatted correctly they the database could be unusable and any automation may not work properly.2 |
| 1. Heed Compiler Warnings | Programmers often overlook heeding compiler warnings which can lead to messy and inefficient code but it still builds. This is a bad habit for programmers to ignore. It is best practice to address all warnings from the compiler. |
| 1. Architect and Design for Security Policies | Following the security policies put in place it of the upmost importance. If security policies are not followed then hackers can infiltrate your systems and steal data from the company or even worse, steal money. |
| 1. Keep It Simple | Overcomplicating code can cause many issues down the road. If are able to make code simple yet effective then anyone can come back in and adjust or make changes. If you code for only yourself then editing your code by another person can lead to issues. |
| 1. Default Deny | This is a security measure taken to remove all access to resources and only allow access to users who need them. Firewalls are a great example of this principle. |
| 1. Adhere to the Principle of Least Privilege | This principle addresses access for users and giving them only the access they need and granting further access or elevated access. |
| 1. Sanitize Data Sent to Other Systems | Cleaning data as it is transferred to other sources is very important way to make sure data is secure and contains no malicious content. Running files through security measures before receiving and sending data can help remove threats. |
| 1. Practice Defense in Depth | Having one layer of defense would cause damage to any system or network if breached. Having a layered system is important and recommended to protect your system. This usually means having a password for logins but also requiring a two form authentication practice. Those accounts with admin privileges may even have a revolving password that changes daily. |
| 1. Use Effective Quality Assurance Techniques | Utilizing quality assurance will make sure your code is formatted and follows any company standards. These company standards have been put in place by senior level personel. |
| 1. Adopt a Secure Coding Standard | It is important to adopt a secure coding standard so that the entire firm/company can code with the some security level and requirements. This can ensure your work is safe from outside sources. |

### 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] | This rule is in place to help developers avoid subtle bugs and undefined behavior arising from differences between C and C++ when using variadic functions. |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, the object passed to va\_start() will undergo a default argument promotion, which results in undefined behavior. |
| #include <cstdarg>    extern "C" void f(float a, ...) {    va\_list list;    va\_start(list, a);    // ...    va\_end(list);  } |

| **Compliant Code** |
| --- |
| In this compliant solution, f() accepts a double instead of a float. |
| #include <cstdarg>    extern "C" void f(double a, ...) {    va\_list list;    va\_start(list, a);    // ...    va\_end(list);  } |

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

| **Principles(s):** 1 & 4. Validate all data types and while keeping the code simple and easy to read. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Clang | 3.9 | -Wvarargs | Does not catch the violation in the third noncompliant code example (it is conditionally supported by Clang) |
| CodeSonar | 8.3p0 | BADMACRO.STDARG\_H | Use of <stdarg.h> Feature |
| Helix QAC | 2024.4 | C++3852, C++3853 |  |
| Klocwork | 2024.4 | CERT.VA\_START.TYPE | [Insert text.] |
| Parasoft C/C++test | 2024.2 | CERT\_CPP-EXP58-a | Use macros for variable arguments correctly |
| Polyspace Bug Finder | R2024a | CERT C++: EXP58-CPP | Checks for incorrect data types for second argument of va\_start (rule fully covered) |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | [STD-002-CPP] | This rule enforces **code correctness and predictability** by ensuring that every possible execution path in a value-returning function provides a return value. Failing to do so leads to **undefined behavior**, which can cause serious runtime errors or silent bugs. |

| **Noncompliant Code** |
| --- |
| [Noncompliant description] |
| int absolute\_value(int a) {    if (a < 0) {      return -a;    }  } |

| **Compliant Code** |
| --- |
| [Compliant description] |
| int absolute\_value(int a) {    if (a < 0) {      return -a;    }    return a;  } |

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

| **Principles(s):** 1. You must validate any external data and this principle will ensure code correctness and predictability by ensuring any value will return valid. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Medium | Probable | Medium | P8 | L2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.10 | return-implicit | Fully checked |
| Axivion Bauhaus Suite | 7.2.0 | CertC++-MSC52 |  |
| Clang | 3.9 | -Wreturn-type | Does not catch all instances of this rule, such as function-try-blocks |
| CodeSonar | 8.3p0 | LANG.STRUCT.MRS LANG.STRUCT.NVNR | Missing return statement Non-void noreturn, |
| Helix QAC | 2024.4 | DF2888 |  |
| Klocwork | 2024.4 | FUNCRET.GEN  FUNCRET.IMPLICIT |  |
| LDRA tool suite | 9.7.1 | 2 D, 36 S | Fully implemented |
| Parasoft C/C++test | 2024.2 | CERT\_CPP-MSC52-a | All exit paths from a function, except main(), with non-void return type shall have an explicit return statement with an expression |
| Polyspace Bug Finder | R2024a | CERT C++: MSC52-CPP | Checks for missing return statements (rule partially covered) |
| SonarQube C/C++ Plugin | 4.10 | S935 |  |
| PVS-Studio | 7.35 | V591 |  |
| RuleChecker | 22.10 | return-implicit | Fully checked |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | [STD-003-CPP] | Unchecked string manipulation is a common cause of **buffer overflows**, which can lead to **security vulnerabilities, crashes, and undefined behavior**. Using **safe string-handling techniques**, ensuring **proper buffer sizing**, and adopting **modern C++ string abstractions** (std::string, std::vector<char>) help mitigate these risks. |

| **Noncompliant Code** |
| --- |
| Because the input is unbounded, the following code could lead to a buffer overflow. |
| #include <iostream>    void f() {    char buf[12];    std::cin >> buf;  } |

| **Compliant Code** |
| --- |
| To solve this problem, it may be tempting to use the std::ios\_base::width() method, but there still is a trap, as shown in this noncompliant code example. |
| #include <iostream>    void f() {    char bufOne[12];    char bufTwo[12];    std::cin.width(12);    std::cin >> bufOne;    std::cin >> bufTwo;  } |

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

| **Principles(s):** 4. Keeping the storage large enough to maintain enough space for data and null values works best when the code is kept simple and accurate. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.10 | stream-input-char-array | Partially checked + soundly supported |
| CodeSonar | 8.3p0 | MISC.MEM.NTERM LANG.MEM.BO LANG.MEM.TO | No space for null terminator Buffer overrun Type overrun |
| Helix QAC | 2024.4 | C++5216  DF2835, DF2836, DF2839, |  |
| Klocwork | 2024.4 | NNTS.MIGHT NNTS.TAINTED NNTS.MUST SV.UNBOUND\_STRING\_INPUT.CIN |  |
| LDRA tool suite | 9.7.1 | 489 S, 66 X, 70 X, 71 X | Partially implemented |
| Parasoft C/C++test | 2024.2 | CERT\_CPP-STR50-b CERT\_CPP-STR50-c CERT\_CPP-STR50-e CERT\_CPP-STR50-f CERT\_CPP-STR50-g | Avoid overflow due to reading a not zero terminated string Avoid overflow when writing to a buffer Prevent buffer overflows from tainted data Avoid buffer write overflow from tainted data Do not use the 'char' buffer to store input from 'std::cin' |
| 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. |
| RuleChecker | 22.10 | stream-input-char-array | Partially checked |
| SonarQube C/C++ Plugin | 4.10 | S3519 |  |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | [STD-004-J] | SQL injection is one of the **most dangerous vulnerabilities**, allowing attackers to compromise databases, manipulate records, or escalate privileges. |

| **Noncompliant Code** |
| --- |
| [Noncompliant description] |
| 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** |
| --- |
| [Compliant description] |
| 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):** 3, 8, 9. These principles all revolve around security and protection. |
| --- |

**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 |
| 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 | 2024.4 | SV.DATA.DB SV.SQL SV.SQL.DBSOURCE | Implemented |
| Parasoft Jtest | 2024.2 | 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 |
| SpotBugs | 4.6.0 | SQL\_NONCONSTANT\_STRING\_PASSED\_TO\_EXECUTE SQL\_PREPARED\_STATEMENT\_GENERATED\_FROM\_NONCONSTANT\_STRING | Implemented |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | [STD-005-CPP] | Dangling pointers introduce **serious reliability and security risks**, including **use-after-free vulnerabilities, heap corruption, crashes, and potential exploits**. To mitigate these risks, always **nullify freed pointers, use smart pointers, avoid returning local addresses, and employ proper memory management techniques**. |

| **Noncompliant Code** |
| --- |
| Typically, dynamic memory allocations and deallocations are far removed, making it difficult to recognize and diagnose such problems. |
| #include <new>    struct S {    void f();  };    void g() noexcept(false) {    S \*s = new S;    // ...    delete s;    // ...    s->f();  } |

| **Compliant Code** |
| --- |
| In this compliant solution, the dynamically allocated memory is not deallocated until it is no longer required. |
| #include <new>    struct S {    void f();  };    void g() noexcept(false) {    S \*s = new S;    // ...    s->f();    delete s;  } |

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

| **Principles(s):** 3, 9. Designing for security policies will help keep everything safe while using quality assurance techniques. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.10 | **dangling\_pointer\_use** |  |
| Axivion Bauhaus Suite | 7.2.0 | **CertC++-MEM50** |  |
| Clang | 3.9 | clang-analyzer-cplusplus.NewDelete clang-analyzer-alpha.security.ArrayBoundV2 | Checked by clang-tidy, but does not catch all violations of this rule. |
| CodeSonar | 8.3p0 | **ALLOC.UAF** | Use after free |
| Compass/ROSE |  |  |  |
| Coverity | v7.5.0 | **USE\_AFTER\_FREE** | Can detect the specific instances where memory is deallocated more than once or read/written to the target of a freed pointer |
| Helix QAC | 2024.4 | **C++4303, C++4304** |  |
| Klocwork | 2024.4 | **UFM.DEREF.MIGHT** **UFM.DEREF.MUST** **UFM.FFM.MIGHT** **UFM.FFM.MUST** **UFM.RETURN.MIGHT** **UFM.RETURN.MUST** **UFM.USE.MIGHT** **UFM.USE.MUST** |  |
| LDRA tool suite | 9.7.1 | **483 S, 484 S** | Partially implemented |
| Parasoft C/C++test | 2024.2 | CERT\_CPP-MEM50-a | Do not use resources that have been freed |
| Parasoft Insure++ |  |  | Runtime detection |
| Polyspace Bug Finder | R2024a | CERT C++: MEM50-CPP | Checks for:   * Pointer access out of bounds * Deallocation of previously deallocated pointer * Use of previously freed pointer   Rule partially covered. |
| PVS-Studio | 7.35 | V586, V774 |  |
| Splint | 5.0 |  |  |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | [STD-006-C] | Assertions are a powerful tool for debugging but should not be relied upon for handling runtime errors in deployed applications. |

| **Noncompliant Code** |
| --- |
| This noncompliant code example uses the assert() macro to verify that memory allocation succeeded. Because memory availability depends on the overall state of the system and can become exhausted at any point during a process lifetime, a robust program must be prepared to gracefully handle and recover from its exhaustion. |
| char \*dupstring(const char \*c\_str) {    size\_t len;    char \*dup;      len = strlen(c\_str);    dup = (char \*)malloc(len + 1);    assert(NULL != dup);      memcpy(dup, c\_str, len + 1);    return dup;  } |

| **Compliant Code** |
| --- |
| This compliant solution demonstrates how to detect and handle possible memory exhaustion: |
| char \*dupstring(const char \*c\_str) {    size\_t len;    char \*dup;      len = strlen(c\_str);    dup = (char\*)malloc(len + 1);    /\* Detect and handle memory allocation error \*/    if (NULL == dup) {        return NULL;    }      memcpy(dup, c\_str, len + 1);    return dup;  } |

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

| **Principles(s):** 1, 9. Validating input data into variables within your code will help keep your debugging cleaner and using effective quality assurance and not relying soley on debugging runtime errors. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 8.3p0 | LANG.FUNCS.ASSERTS | Not enough assertions |
| Coverity | 2017.07 | ASSERT\_SIDE\_EFFECT | Can detect the specific instance where assertion contains an operation/function call that may have a side effect |
| Parasoft C/C++test | 2024.2 | CERT\_C-MSC11-a | Assert liberally to document internal assumptions and invariants |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | [STD-007-CPP] | Proper error handling in C++ revolves around maintaining the correctness, stability, and reliability of a program, especially in the presence of unexpected failures. |

| **Noncompliant Code** |
| --- |
| The implicit invariants of the class are that the array member is a valid (possibly null) pointer and that the nElems member stores the number of elements in the array pointed to by array. |
| #include <cstring>    class IntArray {    int \*array;    std::size\_t nElems;  public:    // ...      ~IntArray() {      delete[] array;    }        IntArray(const IntArray& that); // nontrivial copy constructor    IntArray& operator=(const IntArray &rhs) {      if (this != &rhs) {        delete[] array;        array = nullptr;        nElems = rhs.nElems;        if (nElems) {          array = new int[nElems];          std::memcpy(array, rhs.array, nElems \* sizeof(\*array));        }      }      return \*this;    }      // ...  }; |

| **Compliant Code** |
| --- |
| The function allocates new storage for the copy before changing the state of the object. |
| #include <cstring>    class IntArray {    int \*array;    std::size\_t nElems;  public:    // ...      ~IntArray() {      delete[] array;    }      IntArray(const IntArray& that); // nontrivial copy constructor      IntArray& operator=(const IntArray &rhs) {      int \*tmp = nullptr;      if (rhs.nElems) {        tmp = new int[rhs.nElems];        std::memcpy(tmp, rhs.array, rhs.nElems \* sizeof(\*array));      }      delete[] array;      array = tmp;      nElems = rhs.nElems;      return \*this;    }      // ...  }; |

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

| **Principles(s):** 4, 2. Keeping your error handling simple and precise can help keep you clean code. Do not over think its handling and start at the beginning. Keep an eye on your compiler warnings while coding. It can help keep you up to date instead of waiting to the end. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 8.3p0 | **ALLOC.LEAK** | Leak |
| Helix QAC | 2024.4 | **C++4075, C++4076** |  |
| LDRA tool suite | 9.7.1 | **527 S, 56 D, 71 D** | Partially implemented |
| Parasoft C/C++test | 2024.2 | **CERT\_CPP-ERR56-a** **CERT\_CPP-ERR56-b** | Always catch exceptions Empty 'catch' blocks should not be used |
| Polyspace Bug Finder | R2024a | CERT C++: ERR56-CPP | Checks for exceptions violating class invariant (rule fully covered). |
| PVS-Studio | 7.35 | V565**,**V1023**,**V5002 |  |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Integers** | [STD-008-C] | Signed integer overflow is classified as **undefined behavior**, which means that the C++ standard does not prescribe a specific outcome when it occurs. This allows compilers to optimize code under the assumption that signed overflow will never happen. However, this assumption can lead to unpredictable results, including security vulnerabilities and program instability. |

| **Noncompliant Code** |
| --- |
| [Noncompliant description] |
| void func(signed int si\_a, signed int si\_b) {    signed int sum = si\_a + si\_b;    /\* ... \*/  } |

| **Compliant Code** |
| --- |
| [Compliant description] |
| #include <limits.h>    void f(signed int si\_a, signed int si\_b) {    signed int sum;    if (((si\_b > 0) && (si\_a > (INT\_MAX - si\_b))) ||        ((si\_b < 0) && (si\_a < (INT\_MIN - si\_b)))) {      /\* Handle error \*/    } else {      sum = si\_a + si\_b;    }    /\* ... \*/  } |

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

| **Principles(s):** 1, 2. C++ requires specific input of integers that do not cause an undefined behaviors. This can lead to issues and compiler warnings. Ignoring these can lead to trouble at the end of the project and cause time and possible rework. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 24.04 | integer-overflow | Fully checked |
| CodeSonar | 8.3p0 | ALLOC.SIZE.ADDOFLOW ALLOC.SIZE.IOFLOW ALLOC.SIZE.MULOFLOW ALLOC.SIZE.SUBUFLOW MISC.MEM.SIZE.ADDOFLOW MISC.MEM.SIZE.BAD MISC.MEM.SIZE.MULOFLOW MISC.MEM.SIZE.SUBUFLOW | Addition overflow of allocation size Integer overflow of allocation size Multiplication overflow of allocation size Subtraction underflow of allocation size Addition overflow of size Unreasonable size argument Multiplication overflow of size Subtraction underflow of size |
| Coverity | 2017.07 | TAINTED\_SCALAR  BAD\_SHIFT | Implemented |
| Cppcheck Premium | 24.11.0 | premium-cert-int32-c |  |
| Helix QAC | 2024.4 | C2800, C2860  C++2800, C++2860  DF2801, DF2802, DF2803, DF2861, DF2862, DF2863 |  |
| Klocwork | 2024.4 | NUM.OVERFLOW CWARN.NOEFFECT.OUTOFRANGE NUM.OVERFLOW.DF |  |
| LDRA tool suite | 9.7.1 | 493 S, 494 S | Partially implemented |
| 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 |
| Parasoft Insure++ |  |  | Runtime analysis |
| Polyspace Bug Finder | R2024a | CERT C: Rule INT32-C | Checks for:   * Integer overflow * Tainted division operand * Tainted modulo operand   Rule partially covered. |
| PVS-Studio | 7.35 | V1026, V1070, V1081, V1083, V1085, V5010 |  |
| TrustInSoft Analyzer | 1.38 | signed\_overflow | Exhaustively verified (see one compliant and one non-compliant example). |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Input Output** | [STD-009-CPP] | When using file streams (std::fstream, std::ifstream, std::ofstream), it's essential to follow the standard's rules for mixing input and output operations. The key takeaway from the C++ Standard ([filebuf]) and C Standard (subclause 7.19.5.3) is that improper sequencing of input and output operations can lead to undefined behavior. |

| **Noncompliant Code** |
| --- |
| This noncompliant code example appends data to the end of a file and then reads from the same file. |
| #include <fstream>  #include <string>    void f(const std::string &fileName) {    std::fstream file(fileName);    if (!file.is\_open()) {      // Handle error      return;    }      file << "Output some data";    std::string str;    file >> str;  } |

| **Compliant Code** |
| --- |
| In this compliant solution, the std::basic\_istream<T>::seekg() function is called between the output and input, eliminating the undefined behavior. |
| #include <fstream>  #include <string>    void f(const std::string &fileName) {    std::fstream file(fileName);    if (!file.is\_open()) {      // Handle error      return;    }      file << "Output some data";      std::string str;    file.seekg(0, std::ios::beg);    file >> str;  } |

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

| **Principles(s):** 1, 4, 7. Validating the input of data into the system while keeping the code clean and efficient can help with compiler issues later down the road. Verifying and cleaning the data before it enters and before it leaves your IDE can help the process of data exchange. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion Bauhaus Suite | 7.2.0 | CertC++-FIO50 |  |
| CodeSonar | 8.3p0 | IO.IOWOP  IO.OIWOP | Input After Output Without Positioning  Output After Input Without Positioning |
| Helix QAC | 2024.4 | DF4711, DF4712, DF4713 |  |
| Parasoft C/C++test | 2024.2 | CERT\_CPP-FIO50-a | Do not alternately input and output from a stream without an intervening flush or positioning call |
| Polyspace Bug Finder | R2024a | CERT C++: FIO50-CPP | Checks for alternating input and output from a stream without flush or positioning call (rule fully covered) |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Container** | [STD-010-CPP] | C++ associative containers (std::set, std::map, std::multiset, std::multimap) rely on a strict weak ordering property for their comparison predicates. This requirement ensures that elements are stored, retrieved, and compared in a consistent and well-defined manner. |

| **Noncompliant Code** |
| --- |
| The objects stored in the std::set have an overloaded operator< implementation, allowing the objects to be compared with std::less. |
| #include <iostream>  #include <set>    class S {    int i, j;    public:    S(int i, int j) : i(i), j(j) {}      friend bool operator<(const S &lhs, const S &rhs) {      return lhs.i < rhs.i && lhs.j < rhs.j;    }      friend std::ostream &operator<<(std::ostream &os, const S& o) {      os << "i: " << o.i << ", j: " << o.j;      return os;    }  };    void f() {    std::set<S> t{S(1, 1), S(1, 2), S(2, 1)};    for (auto v : t) {      std::cout << v << std::endl;    }  } |

| **Compliant Code** |
| --- |
| This compliant solution uses std::tie() to properly implement the strict weak ordering operator< predicate. |
| #include <iostream>  #include <set>  #include <tuple>    class S {    int i, j;    public:    S(int i, int j) : i(i), j(j) {}      friend bool operator<(const S &lhs, const S &rhs) {      return std::tie(lhs.i, lhs.j) < std::tie(rhs.i, rhs.j);    }      friend std::ostream &operator<<(std::ostream &os, const S& o) {      os << "i: " << o.i << ", j: " << o.j;      return os;    }  };    void f() {    std::set<S> t{S(1, 1), S(1, 2), S(2, 1)};    for (auto v : t) {      std::cout << v << std::endl;    }  } |

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

| **Principles(s):** 9. By formatting the code properly you are able to follow and make sure the code is stored, retrieved and compared properly using quality assurance and following company standards. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Probable | High | P2 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Helix QAC | 2024.4 | **C++3293** |  |
| Parasoft C/C++test | 2024.2 | **CERT\_CPP-CTR57-a** | For associative containers never use comparison function returning true for equal values |
| Polyspace Bug Finder | R2024a | CERT C++: CTR57-CPP | Checks for predicate lacking strict weak ordering (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.

[Insert your written explanations here.]

### 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 | Medium | Probable | Medium | **P8** | **L2** |
| 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 | High | Likely | High | **P9** | **L2** |
| STD-008-CLG | High | Likely | High | **P9** | **L2** |
| STD-009-CPP | Low | Likely | Medium | **P6** | **L2** |
| STD-010-CPP | Low | Probable | High | **P2** | **L3** |

### Create Policies for Encryption and Triple A

Include all three types of encryption (in flight, at rest, and in use) and each of the three elements of the Triple-A framework using the tables provided***.***

* 1. Explain each type of encryption, how it is used, and why and when the policy applies.
  2. Explain each type of Triple-A framework strategy, how it is used, and why and when the policy applies.

Write policies for each and explain what it is, how it should be applied in practice, and why it should be used.

| 1. **Encryption** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Encryption at rest | Encryption at rest protects stored data by encrypting it to prevent unauthorized access, ensuring security against theft, cyberattacks, and insider threats. It is commonly implemented through full disk encryption, file-level encryption, database encryption, or cloud storage encryption, with secure key management. This encryption is crucial for regulatory compliance and differs from encryption in transit, which secures data during transmission. |
| Encryption in flight | Encryption in flight, also known as encryption in transit, protects data while it is being transmitted over networks to prevent interception and unauthorized access. It is commonly implemented using protocols like TLS/SSL for web traffic, VPNs for secure remote access, and end-to-end encryption for messaging. This ensures data remains confidential and tamper-proof as it moves between systems, devices, or cloud services. |
| Encryption in use | Encryption in use protects data while it is actively being processed, preventing unauthorized access even when loaded in memory. It is implemented through techniques like homomorphic encryption, secure enclaves (e.g., Intel SGX, AMD SEV), and trusted execution environments (TEEs). This ensures sensitive data remains encrypted during computation, reducing risks from insider threats and memory-based attacks. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication verifies a user's identity before granting access to a system, network, or application. It uses methods like passwords, biometrics, security tokens, or multi-factor authentication (MFA) to confirm legitimacy. This step ensures that only authorized individuals can proceed to the next phases. |
| Authorization | Authorization determines what actions or resources a user can access after authentication. It enforces access control policies based on roles, permissions, or attributes, ensuring users can only perform approved operations. This step follows authentication and works alongside accounting to maintain security and compliance. |
| Accounting | Accounting tracks and logs user activities, providing an audit trail for security, compliance, and usage monitoring. It records details such as login times, accessed resources, and actions performed to detect anomalies and enforce policies. This step ensures accountability by maintaining logs for auditing, billing, and forensic investigations. |

**\***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.

| **Rule** | **Reason for Support** |
| --- | --- |
| STD-001-CPP | 1 & 4. Validate all data types and while keeping the code simple and easy to read. |
| STD-002-CPP | 1. You must validate any external data and this principle will ensure code correctness and predictability by ensuring any value will return valid. |
| STD-003-CPP | 4. Keeping the storage large enough to maintain enough space for data and null values works best when the code is kept simple and accurate. |
| STD-004-JAV | 3, 8, 9. These principles all revolve around security and protection. |
| STD-005-CPP | 3, 9. Designing for security policies will help keep everything safe while using quality assurance techniques. |
| STD-006-CLG | 1, 9. Validating input data into variables within your code will help keep your debugging cleaner and using effective quality assurance and not relying soley on debugging runtime errors. |
| STD-007-CPP | 4, 2. Keeping your error handling simple and precise can help keep you clean code. Do not over think its handling and start at the beginning. Keep an eye on your compiler warnings while coding. It can help keep you up to date instead of waiting to the end. |
| STD-008-CLG | 1, 2. C++ requires specific input of integers that do not cause an undefined behaviors. This can lead to issues and compiler warnings. Ignoring these can lead to trouble at the end of the project and cause time and possible rework. |
| STD-009-CPP | 1, 4, 7. Validating the input of data into the system while keeping the code clean and efficient can help with compiler issues later down the road. Verifying and cleaning the data before it enters and before it leaves your IDE can help the process of data exchange. |
| STD-010-CPP | 9. By formatting the code properly you are able to follow and make sure the code is stored, retrieved and compared properly using quality assurance and following company standards. |

**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 |  |
| 2.0 | 02/15/2025 | Updated standards, explanations, and summaries | George Kaline III |  |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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