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



# Green Pace Secure Development Policy

## Contents

[Overview 2](#_77jvcgb4lpz2)

[Purpose 2](#_80rwzfe983yz)

[Scope 2](#_47gwlon7g2ld)

[Module Three Milestone 2](#_u72s1yw39opc)

[Ten Core Security Principles 2](#_kay5qdnyoyts)

[C/C++ Ten Coding Standards 3](#_ml0kaistrfc4)

[Coding Standard 1 4](#_f6okee9p1e5p)

[Coding Standard 2 5](#_ejxsm35o2lmh)

[Coding Standard 3 6](#_ud0ewcz4oinu)

[Coding Standard 4 7](#_cl266am06iei)

[Coding Standard 5 8](#_mdk4jpxkas0p)

[Coding Standard 6 9](#_v0lraydhkgiw)

[Coding Standard 7 10](#_9hg5iha0b28m)

[Coding Standard 8 11](#_lro7qz5itgex)

[Coding Standard 9 13](#_hqs2d0oslb53)

[Coding Standard 10 14](#_sk422tme8nyl)

[Defense-in-Depth Illustration 15](#_fvp320lof3pr)

[Project One 15](#_psb3mxhu8bbm)

[1.](#_jyrctru6hygv) Revise the C/C++ Standards 15

[2.](#_12xcg7pcmh6q) Risk Assessment 15

[3.](#_brywmpcrmscs) Automated Detection 15

[4.](#_8aqewsykqd2u) Automation 15

[5.](#_3k26cg9omcby) Summary of Risk Assessments 16

[6.](#_5la2gv5kkt3l) Create Policies for Encryption and Triple A 16

[7.](#_6lawlednnb2l) Map the Principles 17

[Audit Controls and Management 18](#_h026dti48usl)

[Enforcement 18](#_nkkkpymulqns)

[Exceptions Process 18](#_tntpi3lin3ck)

[Distribution 19](#_sy0lhnnfrfmq)

[Policy Change Control 19](#_4go3sq4kh8ps)

[Policy Version History 19](#_8qkeh35dk9k9)

[Appendix A Lookups 19](#_qe3zep7xc0pz)

[Approved C/C++ Language Acronyms 19](#_4ewcp5jfdkng)

## 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 | Validating input is very important, it helps ensure a system is safe with handling data. It protects the program from SQL injections and cross-site scripting. |
| 1. Heed Compiler Warnings | Warnings should be taken seriously, they can show bugs and security vulnerabilities. Heed warnings can also indicate potential issues like uninitialized variables, unused code, or type mismatches, which could lead to runtime errors if left unresolved. |
| 1. Architect and Design for Security Policies | A very important step because you have to strategically plan for security development, and ensure that the program will still run efficiently. Having detailed plans helps with making sure everyone understands what needs to be done. |
| 1. Keep It Simple | Don’t have the code everywhere and keep it clean, add comments in the code and format it right. Having readable code is very important for going back and debugging code. |
| 1. Default Deny | It blocks everything and only lets selected connections through. This is the best way to ensure only trusted connections can get through. |
| 1. Adhere to the Principle of Least Privilege | Giving the right users read, write, or execute privileges. This is important to ensure the right users have executed because it can have unwanted users messing with the code. |
| 1. Sanitize Data Sent to Other Systems | This has data to be secured before sending. You have to remove crucial data you don't want to be exposed. |
| 1. Practice Defense in Depth | Practicing defence in depth is important because it ensures that there are layers of security getting added to the program. |
| 1. Use Effective Quality Assurance Techniques | Techniques to use can be unit testing, which ensures the program is working right. Another technique can be penetration testing which exposes weaknesses. Finally updating the application to ensure it’s life cycle stays relevant. |
| 1. Adopt a Secure Coding Standard | A good secure coding standard to adopt is University of Michigan Secure Coding and Application Security standard. Some requirements are using a test environment, implementing two-factor authorization, and having proper error handling techniques. |

### 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] | Never qualify a reference type with const or volatile |

| **Noncompliant Code** |
| --- |
| A const-qualified reference to a char is formed instead of a reference to a const-qualified char |
| void f(char c) {  char &const p = c;  p = 'p';  std::cout << c << std::endl;  } |

| **Compliant Code** |
| --- |
| This compliant solution removes the const qualifier |
| void f(char c) {  char &p = c;  p = 'p';  std::cout << c << std::endl;  } |

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

| **Principles(s):**   1. ValidateInput Data: We are ensuring that the data type is referenced right. We have the p correctly called and equaled to c. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Parasoft C/C++test | 2024.2 | CERT\_CPP-DCL52-a | Never qualify a reference type with 'const' or 'volatile' |
| Polyspace Bug Finder | R2024b | CERT C++: DCL52-CPP | Looks for const-qualified reference types |
| Clang | 3.9 | -analyzer-checker=cplusplus | Clang checks for violations of this rule and produces an error without the need to specify any special flags or options |
| Helix QAC | 2025.1 | C++0014 | Ensures a reference type is correctly called |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | [STD-002-CPP] | Do not cast or delete pointers to incomplete classes |

| **Noncompliant Code** |
| --- |
| A class attempts to implement the impl idiom but deletes a pointer to an incomplete class type, resulting in undefined behavior if Body has a nontrivial destructor. |
| class Handle {  class Body \*impl; // Declaration of a pointer to an incomplete class  public:  ~Handle() { delete impl; } // Deletion of pointer to an incomplete class  // ...  }; |

| **Compliant Code** |
| --- |
| The deletion of impl is moved to a part of the code where Body is defined. |
| class Handle {  class Body \*impl; // Declaration of a pointer to an incomplete class  public:  ~Handle();  // ...  };    // Elsewhere  class Body { /\* ... \*/ };    Handle::~Handle() {  delete impl;  } |

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

| **Principles(s):**  2. Heed Compiler Warning: This will lead to a warning as the function deletes an incomplete class type. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Coverity | 6.5 | DELETE\_VOID | Fully implemented |
| CodeSonar | 9.1p0 | Lang.Cast.Pc.Incparse.Doic | Conversion: pointer to incomplete  delete of incomplete class |
| Parasoft C/C++ test | 2024.2 | CERT\_CPP-EXP57-a  CERT\_CPP-EXP57-b | Do not delete objects with incomplete class at the point of deletion  Conversions shall not be performed between a pointer to an incomplete type and any other type |
| Polyspace Bug Finder | R2024b | CERT C++:EXP57-CPP | Checks for conversion or deletion of incomplete class pointer |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | [STD-003-CPP] | Use valid references, pointers, and iterators to reference elements of a basic\_string |

| **Noncompliant Code** |
| --- |
| data is invalidated after the call to replace(), and so its use in g() is undefined behavior. |
| #include <iostream>  #include <string>    extern void g(const char \*);    void f(std::string &exampleString) {  const char \*data = exampleString.data();  // ...  exampleString.replace(0, 2, "bb");  // ...  g(data);  } |

| **Compliant Code** |
| --- |
| The pointer to exampleString's internal buffer is not generated until after the modification from replace() has completed. |
| #include <iostream>  #include <string>    extern void g(const char \*);    void f(std::string &exampleString) {  // ...  exampleString.replace(0, 2, "bb");  // ...  g(exampleString.data());  } |

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

| **Principles(s):**  2. Heed Compiler Warnings: The data in the replace() function is wrong and just uses the g() function which is undefined behavior.  10. Adopt a Secure Coding Standard: Having a good coding standard would help avoid mistakes like this to ensure the program is running smoothly. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 9.1p0 | ALLOC.UAF | Use After Free |
| Helix QAC | 2025.2 | DF4746, DF4747, DF4748, DF4749 | Validates points to basic strings |
| Polyspace Bug Finder | R2024b | CERT C++: STR52-CPP | Checks for use of invalid string iterator (rule partially covered). |
| [Insert text.] | 6.02 | C24 | Fully implemented |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | [STD-004-JAV] | Prevent SQL injection |

| **Noncompliant Code** |
| --- |
| The java.sql.PreparedStatement class properly escapes input strings, preventing SQL injection when used correctly. This code example modifies the doPrivilegedAction() method to use a PreparedStatement instead of java.sql.Statement. However, the prepared statement still permits a SQL injection attack by incorporating the unsanitized input argument username into the prepared statement. |
| import java.sql.Connection;  import java.sql.DriverManager;  import java.sql.ResultSet;  import java.sql.SQLException;  import java.sql.Statement;    class Login {  public Connection getConnection() throws SQLException {  DriverManager.registerDriver(new  com.microsoft.sqlserver.jdbc.SQLServerDriver());  String dbConnection =  PropertyManager.getProperty("db.connection");  // Can hold some value like  // "jdbc:microsoft:sqlserver://<HOST>:1433,<UID>,<PWD>"  return DriverManager.getConnection(dbConnection);  }    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** |
| --- |
| This code validates the length of the username argument, preventing an attacker from submitting an arbitrarily long user name. |
| 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):**   1. Validate Input Data: This validates the users input to ensure no attack happens on the program   8. Practice Defence in Depth: Having more layer of security will make it hard for users accounts getting hacked into  10. Adopt a Secure Coding Standard: Having a good coding standard will ensure that security is added throughout the program. |
| --- |

**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 | 9.0p0 | JAVA.IO.INJ.SQL | SQL injection |
| Parasoft  Jtest | 2024.2 | CERT.IDS00.TDSQL | Protect against SQL injection |
| SonarQube | 9.9 | S2077 and S3649 | Executing SQL queries is security-sensitive |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | [STD-005-C++] | Honor replacement dynamic storage management requirements |

| **Noncompliant Code** |
| --- |
| The global operator new(std::size\_t) function is replaced by a custom implementation. However, the custom implementation fails to honor the behavior required by the function it replaces |
| #include <new>    void \*operator new(std::size\_t size) {  extern void \*alloc\_mem(std::size\_t); // Implemented elsewhere; may return nullptr  return alloc\_mem(size);  }    void operator delete(void \*ptr) noexcept; // Defined elsewhere  void operator delete(void \*ptr, std::size\_t) noexcept; // Defined elsewhere |

| **Compliant Code** |
| --- |
| This compliant solution implements the required behavior for the replaced global allocator function by properly throwing a std::bad\_alloc exception when the allocation fails. |
| #include <new>    void \*operator new(std::size\_t size) {  extern void \*alloc\_mem(std::size\_t); // Implemented elsewhere; may return nullptr  if (void \*ret = alloc\_mem(size)) {  return ret;  }  throw std::bad\_alloc();  }    void operator delete(void \*ptr) noexcept; // Defined elsewhere  void operator delete(void \*ptr, std::size\_t) noexcept; // Defined elsewhere |

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

| **Principles(s):**  3. Architect and Design for Security Policies: Having the memory allocate data right is important to having a secure and efficient program. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Helix QAC | 2025.2 | DF4736, DF4737, DF4738, DF4739 | Makes sure the dynamic storage meets requirements |
| Klocwork | 2025.2 | CERT.MEM.OVERRIDE.DELETE  CERT.MEM.OVERRIDE.NEW | Ensures storage requirements |
| Parasoft C/C++test | 2024.2 | CERT\_CPP-MEM55-a | The user defined 'new' operator should throw the 'std::bad\_alloc' exception when the allocation fails |
| Polyspace Bug Finder | R2024b | CERT C++: MEM55-CPP | Checks for replacement allocation/deallocation functions that do not meet requirements of the Standard (rule fully covered) |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | [STD-006-C++] | Use a static assertion to test the value of a constant expression |

| **Noncompliant Code** |
| --- |
| the assert() macro to assert a property concerning a memory-mapped structure that is essential for the code to behave correctly. |
| #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** |
| --- |
| This portable compliant solution uses static\_assert. |
| #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):**  9. Use Effective Quality Assurance Techniques: Testing the programs right is very crucial to having no bugs in the program. For this we use static assertion to test the value of a constant expression. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Clang | 3.9 | misc-static-assert | Checked by clang-tidy |
| CodeSonar | 9.1p0 | customization | Users can implement a custom check that reports uses of the assert() macro |
| LDRA tool suite | 9.71 | 44 S | Fully implemented |
| ECLAIR | 1.2 | CC2.DCL03 | Fully implemented |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | [STD-007-C++] | Do not leak resources when handling exceptions |

| **Noncompliant Code** |
| --- |
| pst is not properly released when process\_item throws an exception, causing a resource leak. |
| #include <new>    struct SomeType {  SomeType() noexcept; // Performs nontrivial initialization.  ~SomeType(); // Performs nontrivial finalization.  void process\_item() noexcept(false);  };    void f() {  SomeType \*pst = new (std::nothrow) SomeType();  if (!pst) {  // Handle error  return;  }    try {  pst->process\_item();  } catch (...) {  // Process error, but do not recover from it; rethrow.  throw;  }  delete pst;  } |

| **Compliant Code** |
| --- |
| The exception handler frees pst by calling delete. |
| #include <new>    struct SomeType {  SomeType() noexcept; // Performs nontrivial initialization.  ~SomeType(); // Performs nontrivial finalization.    void process\_item() noexcept(false);  };    void f() {  SomeType \*pst = new (std::nothrow) SomeType();  if (!pst) {  // Handle error  return;  }  try {  pst->process\_item();  } catch (...) {  // Process error, but do not recover from it; rethrow.  delete pst;  throw;  }  delete pst;  } |

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

| **Principles(s):**  2. Heed Compiler Warnings: The exception causes a resource leak which is big security issue.  9. Use Effective Quality Assurance Techniques: We should use more testing to ensure the program does not have any data leaks.  10. Adopt a Secure Coding Standard: Having a code standard helps handle errors. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 9.1p0 | ALLOC.LEAK | Leak |
| Helix QAC | 2025.2 | DF4756, DF4757, DF4758 | Handles exceptions |
| Parasoft C/C++test | 2024.2 | CERT\_CPP-ERR57-a | Ensure resources are freed |
| Polyspace Bug Finder | R2024b | CERT C++: ERR57-CPP | Checks for resource leaks caused by exception |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Object-Oriented-Programming | [STD-008-C++] | Do not use pointer-to-member operators to access nonexistent members |

| **Noncompliant Code** |
| --- |
| A pointer-to-member object is obtained from D::g but is then upcast to be a B::\*. When called on an object whose dynamic type is D, the pointer-to-member call is well defined. However, the dynamic type of the underlying object is B, which results in undefined behavior |
| struct B {  virtual ~B() = default;  };    struct D : B {  virtual ~D() = default;  virtual void g() { /\* ... \*/ }  };    void f() {  B \*b = new B;    // ...    void (B::\*gptr)() = static\_cast<void(B::\*)()>(&D::g);  (b->\*gptr)();  delete b;  } |

| **Compliant Code** |
| --- |
| The upcast is removed, rendering the initial code ill-formed and emphasizing the underlying problem that B::g() does not exist. This compliant solution assumes that the programmer's intention was to use the correct dynamic type for the underlying object. |
| struct B {  virtual ~B() = default;  };    struct D : B {  virtual ~D() = default;  virtual void g() { /\* ... \*/ }  };    void f() {  B \*b = new D; // Corrected the dynamic object type.    // ...  void (D::\*gptr)() = &D::g; // Moved static\_cast to the next line.  (static\_cast<D \*>(b)->\*gptr)();  delete b;  } |

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

| **Principles(s):**  2. Heed Compiler Warnings: This can make the programs behavior to be unpredictable and could lead to crashes, incorrect results, or security vulnerabilities |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 9.1p0 | LANG.MEM.UVAR | Uninitialized Variable |
| Parasoft C/C++test | 2024.2 | CERT\_CPP-OOP55-a | A cast shall not convert a pointer to a function to any other pointer type, including a pointer to function type |
| Parasoft Insure++ | 2024.2 | CERT\_CPP-OOP55-a | Runtime detection |
| Polyspace Bug Finder | R2024b | CERT C++: OOP55-CPP | Checks for pointers to member accessing non-existent class members (rule fully covered). |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Error Handling | [STD-009-C++] | Detect errors when converting a string to a number |

| **Noncompliant Code** |
| --- |
| Multiple numeric values are converted from the standard input stream. However, if the text received from the standard input stream cannot be converted into a numeric value that can be represented by an int, the resulting value stored into the variables i and j may be unexpected. |
| #include <iostream>    void f() {  int i, j;  std::cin >> i >> j;  // ...  } |

| **Compliant Code** |
| --- |
| Exceptions are enabled so that any conversion failure results in an exception being thrown. However, this approach cannot distinguish between which values are valid and which values are invalid and must assume that all values are invalid. Both the badbit and failbit flags are set to ensure that conversion errors as well as loss of integrity with the stream are treated as exceptions. |
| #include <iostream>    void f() {  int i, j;    std::cin.exceptions(std::istream::failbit | std::istream::badbit);  try {  std::cin >> i >> j;  // ...  } catch (std::istream::failure &E) {  // Handle error  }  } |

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

| **Principles(s):**   1. Heed Compiler Warnings: An error will pop up if the handling doesn’t catch it.   9. Use Effective Quality Assurance Techniques: Having a good quality assurance technique will ensure the error handling will catch string to int errors.  10. Adopt a Secure Coding Standard: Having a secure coding standard will ensure error handling is done right. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Clang | 3.9 | cert-err34-c | Checked by clang-tidy; only identifies use of unsafe C Standard Library functions corresponding to ERR34-C |
| Helix QAC | 2025.2 | C++3161 | Detects errors when a string is converted to a number |
| Klocwork | 2025.2 | CERT.ERR.CONV.STR\_TO\_NUM | Ensures errors are detected when converting a string to a number |
| Polyspace Bug Finder | R2024b | CERT C++: ERR62-CPP | Checks for unvalidated string-to-number conversion (rule fully covered) |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Input Output (FIO) | [STD-010-C++] | Do not alternately input and output from a file stream without an intervening positioning call |

| **Noncompliant Code** |
| --- |
| There is no intervening positioning call between the formatted output and input calls, the behavior is undefined. |
| #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 |

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

| **Principles(s):**   1. Validate Input Data: If the data is not correctly set up the program can write or read from incorrect positions, leading to data corruption. Also can lead to crashes. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 9.1p0 | IO.IOWOP and IO.OIWOP | Input After Output Without Positioning.  Output After Input Without Positioning. |
| Helix QAC | 2025.2 | DF4711, DF4712, DF4713 | Ensures input and output are not alternately from a file stream. |
| 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 | R2024b | CERT C++: FIO50-CPP | Checks for alternating input and output from a stream without flush or positioning call (rule fully 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.

For automation we need to add it in the pre-production, specify the design process. We need to clarify where we want to automate data and how to make sure we are secure about the automation. Having automation in the design process the team knows what the program will function like. Also we would need automation in the verify and testing part of pre-production. This will help make testing faster and ensure we don’t miss little details.

### 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 | P3 | L3 |
| STD-002-CPP | Medium | Unlikely | Low | P4 | L3 |
| STD-003-CPP | High | Probable | High | P6 | L2 |
| STD-004-Jav | High | Like;y | Medium | P18 | L1 |
| STD-005-CPP | High | Likely | High | P9 | L2 |
| STD-006-CPP | Low | Unlikely | Low | P3 | L3 |
| STD-007-CPP | Low | Probable | Low | P2 | L3 |
| STD-008-CPP | High | Probable | Low | P6 | L2 |
| STD-009-CPP | Medium | Unlikely | Medium | P4 | L3 |
| STD-010-CPP | Low | Likely | Low | P6 | L2 |

### 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 | This is when data is stored on a ssd or hard drive and isn’t being touched. The data is still protected by encryption to ensure there is no unauthorized access. |
| Encryption in flight | This is the process of encrypting data while it is being transmitted over a network. This ensures that even if intercepted, the data remains unreadable without the proper decryption key. It is very important to keep the data secured when transmitting over a network because you never know when someone will attack. |
| Encryption in use | This allows the data to be used, while still having encryption. This is great because it protects sensitive data from unauthorized access and modification during use. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | This is very important because it verifies the user's account. It ensures nobody can get in unless they have the right password and username. |
| Authorization | Authorization determines what actions they are allowed to perform on the network. It defines the specific resources and services a user can access and what they can do with them, based on their assigned permissions |
| Accounting | This component tracks and records user activity on the network. It logs user sessions, login times, resource usage, and actions taken, for auditing, billing, and troubleshooting purposes. |

**\***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 |  |
| 2.0 | 08/08/2025 | Security Policies Instituted | Eric Diaz |  |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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