**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 all inputs are rigorously checked against expected formats. Invalid or malicious data must be filtered out to protect against vulnerabilities. |
| 1. Heed Compiler Warnings | Compiler and static analyzer warnings often highlight potential issues in the codebase. Addressing these warnings proactively can help preempt runtime errors and security risks. |
| 1. Architect and Design for Security Policies | Incorporate security considerations into the architecture and design phases so as to minimize the impact of vulnerabilities throughout the entire system |
| 1. Keep It Simple | Favor keeping implementations/solutions simple so as to reduce the likelihood of introducing bugs during development. |
| 1. Default Deny | Deny all access by default and grant access to resources as necessary |
| 1. Adhere to the Principle of Least Privilege | Restrict privileges of users so that they have the minimum access necessary to preform their designated functions/goals. |
| 1. Sanitize Data Sent to Other Systems | Validate and sanitize data before transmitting it to other systems. |
| 1. Practice Defense in Depth | Implement a security strategy that contains multiple layers of defense so that if a layer fails or a threat actor bypasses a layer, the other layers will remain to defend or mitigate the potential threat |
| 1. Use Effective Quality Assurance Techniques | Integrate security into the QA process through techniques like static analysis, fuzz testing, and code audits to discover and remediate flaws early. |
| 1. Adopt a Secure Coding Standard | Consistently apply a secure coding standard tailored to the programming language and platform in use to reduce the introduction of known vulnerabilities |

### C/C++ Ten Coding Standards

Complete the coding standards portion of the template according to the Module Three milestone requirements. In Project One, follow the instructions to add a layer of security to the existing coding standards. Please start each standard on a new page, as they may take up more than one page. The first seven coding standards are labeled by category. The last three are blank so you may choose three additional standards. Be sure to label them by category and give them a sequential number for that category. Add compliant and noncompliant sections as needed to each coding standard.

#### Coding Standard 1

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Type** | [STD-001-CPP] | Obey the one-definition rule |

| **Noncompliant Code** |
| --- |
| In C++ a translation unit consists of an implementation file and all the headers that it includes directly or indirectly. Two different translation units define a class of the same name with differing definitions. Although functionally equivalent they are not defined using the same sequence of tokens, resulting in undefined behavior. |
| // a.cpp  Struct S {  int a;  };  //b.cpp  class S {  int a;  }; |

| **Compliant Code** |
| --- |
| A single shared header file is used to define the struct, ensuring consistency of the object across translation units. |
| //S.h  Struct S {  Int a;  };  //a.cpp  #include “S.h”  //b.cpp  #include “S.h” |

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

| **Principles(s):**  Architect and Design for Security Policies - Promotes consistency and predictability by establishing shared design components  Keep It Simple - Reduces the chance of errors by consolidating definitions into one file.  Adopt a Secure Coding Standard - Aligns with best practices for maintainable and portable C++ code. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 20.10 | type-compatibility  definition-duplicate  undefined-extern  undefined-extern-pure-virtual  external-file-spreading  type-file-spreading | Partially checked |
| Axivion Bauhaus Suite | 7.2.0 | CertC++-DCL60 | - |
| Parasoft C/C++test | 2021.1 | CERT\_CPP-DCL60-a | A class, union or enum name (including qualification, if any) shall be a unique identifier |
| LDRA tool suite | 9.7.1 | 286 S, 287 S | Fully Implemented |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | [STD-002-CPP] | Do not read uninitialized memory |

| **Noncompliant Code** |
| --- |
| An uninitialized local variable i is evaluated as part of the expression to print its value |
| #include <iostream>  void f() {  int i;  std::cout << i;  } |

| **Compliant Code** |
| --- |
| Prior to printing the value of variable i, it is initialized with value 0 |
| #include <iostream>  void f() {  int i = 0;  std::cout << i;  } |

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

| **Principles(s):**  Keep It Simple - The fix requires minimal effort, a single line of initialization.  Adopt a Secure Coding Standard - Follows known best practices to avoid undefined behavior and bugs |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 20.10 | uninitialized read | Partially checked |
| Helix QAC | 2021.2 | C++2726, C++2727, C++2728, C++2961, C++2962, C++2963, C++2966, C++2967, C++2968, C++2971, C++2972, C++2973, C++2976, C++2977, C++2978 | - |
| LDRA tool suite | 9.7.1 | 53 D, 69 D, 631 S, 652 S | Partially implemented |
| Polyspace Bug Finder | R2021a | CERT C++: EXP53-CPP | Checks for:  ⦁ Non-initialized variable  ⦁ Non-initialized pointer  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. |

| **Noncompliant Code** |
| --- |
| A std::string object is created from the results of a call to std::getenv(), but when the environment variable does not exist the call returns a null pointer which leads to undefined behavior |
| #include <cstdlib> #include <string>   void f() {  std::string tmp(std::getenv("TMP"));  if (!tmp.empty()) {  // ...  } } |

| **Compliant Code** |
| --- |
| The contents of tmp are checked for null before constructing the std::string object |
| #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):**  **Validate Input Data -** Checks for null before using the input  **Heed Compiler Warnings -** Prevents runtime crashes flagged by static analysis tools.  **Keep It Simple -** Minimal logic added to prevent undefined behavior.  **Adopt a Secure Coding Standard -** Follows best practices to avoid unsafe pointer usage. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 20.10 | Assert\_failure | - |
| Helix QAC | 2021.2 | C++4770, C++4771, C++4772, C++4773, C++4774 | - |
| Klocwork | 2021.1 | NPD.CHECK.CALL.MIGHT  NPD.CHECK.CALL.MUST  NPD.CHECK.MIGHT    NPD.CHECK.MUST    NPD.CONST.CALL  NPD.CONST.DEREF  NPD.FUNC.CALL.MIGHT  NPD.FUNC.CALL.MUST  NPD.FUNC.MIGHT  NPD.FUNC.MUST    NPD.GEN.CALL.MIGHT  NPD.GEN.CALL.MUST  NPD.GEN.MIGHT  NPD.GEN.MUST    RNPD.CALL  RNPD.DEREF | - |
| Parasoft C/C++test | 2021.1 | CERT\_CPP-STR51-a | Avoid null pointer dereferencing |

#### Coding Standard 4

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

| **Noncompliant Code** |
| --- |
| The untrusted input of username is embedded within the SQL query string and is not parameterized or sanitized properly. |
| 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** |
| --- |
| The SQL query uses parameter placeholders for arguements username and password and is properly inserted using the setString() function of the object stmt of type PreparedStatement |
| 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):**  Validate Input Data - Checks for acceptable input length and format  Adhere to the Principle of Least Privilege - Prevents unauthorized access by enforcing proper access controls  Sanitize Data Sent to Other Systems - Prevents injection attacks by not trusting user-supplied input to run on a database  Adopt a Secure Coding Standard - Follows modern practices for secure database interaction |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Coverity | 7.5 | SQLI FB.SQL\_PREPARED\_STATEMENT\_GENERATED\_ FB.SQL\_NONCONSTANT\_STRING\_PASSED\_TO\_EXECUTE | Implemented |
| The Checker Framework | 2.1.3 | Tainting Checker | Trust and security errors (see Chapter 8) |
| Fortify | 1.0 | HTTP\_Response\_Splitting SQL\_Injection\_\_Persistence SQL\_Injection | Implemented |
| Parasoft Jtest | 2021.1 | CERT.IDS00.TDSQL | Protect against SQL injection |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | [STD-005-CPP] | Do not access freed memory |

| **Noncompliant Code** |
| --- |
| After deleting the memory allocated for s of type S, it is dereferenced causing undefined behavior |
| #include <new>   struct S {  void f(); };   void g() noexcept(false) {  S \*s = new S;  // ...  delete s;  // ...  s->f(); } |

| **Compliant Code** |
| --- |
| Memory is deallocated (freed) only after object s of type S is not needed anymore. |
| #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):**  Heed Compiler Warnings – compilers and other tools will flag use-after-free issues; fix them before runtime  Keep It Simple - Clear allocation and deallocation flow makes logic safer and easier to understand  Adopt a Secure Coding Standard - Prevents undefined behavior and memory corruption. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| 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. |
| 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 |
| Parasoft C/C++test | 2021.1 | CERT\_CPP-MEM50-a | Do not use resources that have been freed |
| Parasoft Insure++ | - | - | Runtime detection |

#### 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 |

| **Noncompliant Code** |
| --- |
| The assert() function is being used in this code to determine the size of the timer struct at runtime and only if the function is executed. Depending on the complier alignment rules the program could break |
| #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** |
| --- |
| Instead of determining the size of the struct at runtime, the static\_assert() function will fail at compile time, producing a meaningful error close to where the bug would originate |
| #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):**  Heed Complier Warnings - Compiler will notify immediately if the condition fails  Adopt a Secure Coding Standard - Promotes compile-time verification of memory layout to prevent subtle bugs |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion Bauhaus Suite | 7.2.0 | CertC-DCL03 | - |
| Clang | 3.9 | misc-static-assert | Checked by clang-tidy |
| ECLAIR | 1.2 | CC2.DCL03 | Fully implemented |
| LDRA tool suite | 9.7.1 | 44 S | Fully implemented |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | [STD-007-CPP] | Do not abruptly terminate the program |

| **Noncompliant Code** |
| --- |
| In the following code f() is set up to run when the program is shutting down. f() calls throwing\_func() in which if it throws an exception will cause a forceful shutdown via std::terminate() |
| #include <cstdlib>   void throwing\_func() noexcept(false);   void f() { // Not invoked by the program except as an exit handler.  throwing\_func(); }   int main() {  if (0 != std::atexit(f)) {  // Handle error  }  // ... } |

| **Compliant Code** |
| --- |
| The following compliant code handles the exception throwing\_func() throws inside of f(). This avoids an implicit call to std::terminate() |
| #include <cstdlib>  void throwing\_func() noexcept(false);  void f() { // Not invoked by the program except as an exit handler.  try { throwing\_func();  } catch (...) {  // Handle error  }  }  int main() {  if (0 != std::atexit(f)) {  // Handle error  }  // ...  } |

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

| **Principles(s):**  Keep It Simple – Defining clear structured exit behavior  Use Effective Quality Assurance Techniques - Exception handling reduces hidden failures  Adopt a Secure Coding Standard - Complies with C++ best practices for exception safety |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 6.1p0 | BADFUNC.ABORT  BADFUNC.EXIT | Use of abort  Use of exit |
| Klocwork | 2021.1 | MISRA.CATCH.ALL  CERT.ERR.ABRUPT\_TERM | - |
| LDRA tool suite | 9.7.1 | 122 S | Enhanced Enforcement |
| Polyspace Bug Finder | R2021a | CERT C++: ERR50-CPP | Checks for implicit call to terminate() function (rule partially covered) |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| String Range | [STD-008-CPP] | Range check element access |

| **Noncompliant Code** |
| --- |
| The unchecked index operator of get\_index() on s may result in undefined behavior if the index is out of bounds of s |
| #include <string>   extern std::size\_t get\_index();   void f() {  std::string s("01234567");  s[get\_index()] = '1'; } |

| **Compliant Code** |
| --- |
| The following compliant code manually checks that the index returned from get\_index() is within the valid range before accessing the string with the std::string::operator[] |
| #include <string>  extern std::size\_t get\_index();  void f() {  std::string s("01234567");  std::size\_t i = get\_index();  if (i < s.length()) {  s[i] = '1';  } else {  // Handle error  } } |

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

| **Principles(s):**  Keep It Simple - Straightforward boundary check avoids complex logic  Adopt a Secure Coding Standard - Prevents common runtime vulnerabilities like buffer overruns |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.10 | **assert\_failure** | [Insert text.] |
| CodeSonar | 9.0p0 | LANG.MEM.BO LANG.MEM.BU LANG.MEM.TBA LANG.MEM.TO LANG.MEM.TU | Buffer overrun Buffer underrun Tainted buffer access Type overrun Type underrun |
| Helix QAC | 2025.1 | C++3162, C++3163, C++3164, C++3165 | - |
| Parasoft C/C++test | 2024.2 | CERT\_CPP-STR53-a | Guarantee that container indices are within the valid range |
| Polyspace Bug Finder | R2024b | CERT C++: STR53-CPP | Checks for:  Array access out of bounds  Array access with tainted index  Pointer dereference with tainted offset  Rule partially covered. |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Input Output | [STD-009-CPP] | Close files when they are no longer needed |

| **Noncompliant Code** |
| --- |
| A file is opened using std::fstream, but then calls std::terminate without explicitly closing the file, leaving the file improperly closed. |
| #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** |
| --- |
| In the following compliant code prior to calling std::terminate(), the file is propely closed by calling file.close(). This allows the file buffer to be properly flushed and resources to be safely released. |
| #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):**  Keep It Simple - Straightforward coding pattern  Adopt a Secure Coding Standard - Supports graceful shutdown |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 9.0p0 | ALLOC.LEAK | Leak |
| Helix QAC | 2025.1 | DF4786, DF4787, DF4788 | - |
| Klocwork | 2025.1 | RH.LEAK | - |
| Parasoft C/C++test | 2024.2 | CERT\_CPP-FIO51-a | Ensure resources are freed |
| Parasoft Insure++ | - | - | Runtime detection |
| Polyspace Bug Finder | R2024b | CERT C++: FIO51-CPP | Checks for resource leak (rule partially covered) |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Object Oriented Programming | [STD-010-CPP] | Write Constructor Member initializers in the canonical order |

| **Noncompliant Code** |
| --- |
| The following noncompliant code has members someVal and dependsOnSomeVal. Given the constructor’s initializer list, someVal would appear to initialize first but in reality the members are initialized by the order that they are declared within the Class definition. This means dependsOnSomeVal is initialized first before someVal, causing the unexpected behavior as it has yet to be defined as a value. |
| class C {  int dependsOnSomeVal;  int someVal;   public:  C(int val) : someVal(val), dependsOnSomeVal(someVal + 1) {} }; |

| **Compliant Code** |
| --- |
| The declaration order has been changed so that someVal comes before dependsOnSomeVal. The following code ensures that the constructor’s member initialization list follows the order they were declared. |
| class C {  int someVal;  int dependsOnSomeVal;   public:  C(int val) : someVal(val), dependsOnSomeVal(someVal + 1) {} }; |

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

| **Principles(s):**  Keep It Simple - Proper member ordering avoids non-obvious bugs  Adopt a Secure Coding Standard - Aligns with C++ best practices for object construction |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 20.10 | initializer-list-order | Fully checked |
| Axivion Bauhaus Suite | 7.2.0 | CertC++-OOP53 | - |
| LDRA tool suite | 9.7.1 | 206 S | Fully implemented |
| Parasoft C/C++test | 2021.1 | CERT\_CPP-OOP53-a | List members in an initialization list in the order in which they are declared |

### 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 | High | Unlikely | Medium | High | 2 |
| **STD-003-CPP** | High | Likely | Medium | High (18) | 1 |
| **STD-004-CPP** | High | Probable | Medium | High (12) | 1 |
| STD-004-JAV | High | Likely | Medium | High (18) | 1 |
| STD-005-CPP | High | Likely | Medium | High (18) | 1 |
| STD-006-CLG | Low | Unlikely | High | Low (1) | 3 |
| **STD-007-CPP** | Low | Probable | Medium | Low (4) | 3 |
| **STD-008-CPP** | High | Unlikely | Medium | Medium (6) | 2 |
| STD-009-CPP | Medium | Unlikely | Medium | Low (4) | 3 |
| **STD-010-CPP** | Medium | Unlikely | Medium | Low (4) | 3 |

### 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 refers to protecting data that is stored on a device or in the cloud. The policy ensures that data is encrypted when it is not actively being used or moved. This helps prevent unauthorized access if the device is lost, stolen, or hacked. Disk encryption tools, secured file systems, and endpoint protection can all help enforce this policy. |
| Encryption in flight | Encryption in transit protects data while it’s being transmitted from one system to another. This can be done using protocols like HTTPS, VPNs, or TLS. Firewalls and authentication methods also help ensure the integrity of the communication. |
| Encryption in use | Encryption in use involves protecting data that is actively being accessed, edited, or processed by applications or users. To enforce this, organizations can implement memory encryption, access controls, and identity verification. It ensures sensitive data stays protected throughout its entire lifecycle. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication is the process of verifying a person's identity before granting access to a system. Common methods include passwords, one-time codes, biometric scans, and security certificates. Strong authentication helps prevent unauthorized access, identity theft, and security breaches by confirming users are who they claim to be |
| Authorization | Authorization defines what a verified user is allowed to do within a system. This includes access to files, applications, tools, and specific actions like reading, writing, or deleting data. This limits access based on user roles and responsibilities, helping reduce the risk of internal misuse or accidental data exposure. It ensures users only interact with data relevant to the tasks related to their assigned role. |
| Accounting | Accounting involves tracking and recording user activity within a system. This includes login times, data accessed, resources used, and other key activities. This provides a record of system activity for security monitoring, auditing, and forensic investigations. It helps organizations detect suspicious behavior, enforce compliance, and maintain accountability. |

**\***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 | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| 1.2 | 6/15/25 | Module 6 Milestone | Anthony Aleman | [Insert text.] |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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