**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 | Validate all input from all sources of data that are untrusted. By properly validating input data you can help and eliminate software vulnerabilities. |
| 1. Heed Compiler Warnings | Code should be compiled by using the top warning accessible for the compiler and eliminate warnings. |
| 1. Architect and Design for Security Policies | Security policies should be enforced by creating a software architecture and design that implements this. |
| 1. Keep It Simple | The overall design and layout should be small and simple. More complex designs could cause more errors. |
| 1. Default Deny | Default access to the software should be denied. This helps to protect access to the software and only allows access when it is permitted. |
| 1. Adhere to the Principle of Least Privilege | There should be a set of privileges that execute when completing a task. This helps to minimize the time an attacker has when gaining access. |
| 1. Sanitize Data Sent to Other Systems | Data should be sanitized when passed to other complex subsystems to help with controlling attackers when they invoke unused components. |
| 1. Practice Defense in Depth | Have multiple defense tactics to help and stop and attack. This can help and make it so that if an attacker makes it through one layer the next one is there to stop them from continuing. |
| 1. Use Effective Quality Assurance Techniques | Eliminating and identifying vulnerabilities can be accomplished by having good quality assurance. By having good testing, it can lead to a more secure system overall. |
| 1. Adopt a Secure Coding Standard | By having a coding standard that is secure you can help and maintain a safe environment for your target development language. |

### 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** | [DLC-50-CPP] | Do not define a C-style variadic function. |

| **Noncompliant Code** |
| --- |
| This noncompliant code example uses a C-style variadic function to add a series of integers together. The function reads arguments until the value 0 is found. Calling this function without passing the value 0 as an argument (after the first two arguments) results in undefined behavior. Furthermore, passing any type other than an int also results in undefined behavior. |
| #include <cstdarg>    **int** add(**int** first, **int** second, ...) {  **int** r = first + second;  **va\_list** va;  **va\_start**(va, second);  **while** (**int** v = **va\_arg**(va, **int**)) {      r += v;    }  **va\_end**(va);  **return** r;  } |

| **Compliant Code** |
| --- |
| In this compliant solution, a variadic function using a function parameter pack is used to implement the add() function, allowing identical behavior for call sites. Unlike the C-style variadic function used in the noncompliant code example, this compliant solution does not result in undefined behavior if the list of parameters is not terminated with 0. Additionally, if any of the values passed to the function are not integers, the code is ill-formed rather than producing undefined behavior. |
| #include <type\_traits>    **template** <**typename** Arg, **typename** std::enable\_if<std::is\_integral<Arg>::value>::type \* = nullptr>  **int** add(Arg f, Arg s) { **return** f + s; }    **template** <**typename** Arg, **typename**... Ts, **typename** std::enable\_if<std::is\_integral<Arg>::value>::type \* = nullptr>  **int** add(Arg f, Ts... rest) {  **return** f + add(rest...);  } |

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

| **Principles(s):** [Name the principle and explain how it maps to this standard.] |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 22.10 | Function-ellipsis | Fully checked |
| Axivion Bauhaus Suite | 7.2.0 | CertC++-DLC50 |  |
| Clang | 3.9 | cert-dlc50-cpp | Checked by clang-tidy |
| CodeSonar | 7.1.p0 | LANG.STRUCT.ELLIPSIS | Ellipsis |
| Helix QAC | 2022.3 | C++2012, C++2625 |  |
| Klocwork | 2022.3 | MIRSA.FUNC.VARAG |  |
| LDRA tool suite | 9.7.1 | 41 S | Fully Implemented |
| Parasoft C/C++test | 2022.1 | CERT\_CPP-DLC50-a | Functions shall not be defined with a variable number of arguments. |
| Polyspace Bug Finder | R2022b | CERT C++:DLC50-CPP | Checks for function definition with ellipsis notation. |
| PRQA QA+C++ | 4.4 | 2012, 2625 |  |
| RuleChecker | 22.10 | Function-ellipsis | Fully checked |
| SonarQube C/C++ Plugin | 4.10 | FunctionEllipsis |  |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | [INT-50-CPP] | Do not cast to an out-of-range enumeration value. |

| **Noncompliant Code** |
| --- |
| This noncompliant code example attempts to check whether a given value is within the range of acceptable enumeration values. However, it is doing so after casting to the enumeration type, which may not be able to represent the given integer value. On a two's complement system, the valid range of values that can be represented by EnumType are [0..3], so if a value outside of that range were passed to f(), the cast to EnumType would result in an unspecified value, and using that value within the if statement results in unspecified behavior. |
| **enum** EnumType {    First,    Second,    Third  };    **void** f(**int** intVar) {    EnumType enumVar = **static\_cast**<EnumType>(intVar);    **if** (enumVar < First || enumVar > Third) {      // Handle error    }  } |

| **Compliant Code** |
| --- |
| This compliant solution checks that the value can be represented by the enumeration type before performing the conversion to guarantee the conversion does not result in an unspecified value. It does this by restricting the converted value to one for which there is a specific enumerator value. |
| **enum** EnumType {  First,    Second,    Third  };    **void** f(**int** intVar) {  **if** (intVar < First || intVar > Third) {      // Handle error    }    EnumType enumVar = **static\_cast**<EnumType>(intVar);  } |
|  |

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

| **Principles(s):** [Name the principle and explain how it maps to this standard.] |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astee | 22.10 | cast-integer-to-enum | Partially checked |
| Axivion Bauhaus Suite | 7.2.0 | CertC++-INT50 |  |
| CodeSonar | 7.1p0 | LANG.CAST.COERCE  LANG.CAST.VALUE | Coercion Alters Value  Cast Alters Value |
| Helix QAC | 2022.3 | C++3013 |  |
| Parasoft C/C++test | 2022.1 | CERT\_CPP-INT50-a | An expression with enum underlying type shall only have values corresponding to the enumerators of the enuymeration. |
| PRQA QA=C++ | 4.4 | 3013 |  |
| PVS-Studio | 7.22 | V1016 |  |
| RuleChecker | 22.10 | cast-integer-to-enum | Partially checked |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | [STR-50-CPP] | Guarantee that storage for strings has sufficient space for character data and the null terminator. |

| **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** |
| --- |
| The best solution for ensuring that data is not truncated and for guarding against buffer overflows is to use std::string instead of a bounded array, as in this compliant solution. |
| #include <iostream>  #include <string>    **void** f() {    std::string input;    std::string stringOne, stringTwo;    std::cin >> stringOne >> stringTwo;  } |

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

| **Principles(s):** [Name the principle and explain how it maps to this standard.] |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astee | 22.10 | stream-input-ch-array | Partially checked + soundly supported |
| CodeSonar | 7.1p0 | MISC.MEM.NTERM  LANG.MEM.BO  LANG.MEM.TO | No space for null terminator  Buffer overrun  Type overrun |
| Helix QAC | 2022.3 | C++2835, C++2836, C++2839, C++5216 |  |
| Klocwork | 2022.3 | 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 | 2022.1 | CERT\_CPP-STR50-b  CERT\_CPP-STR50-c  CERT\_CPP-STR50-e  CERT\_CPP-STR50-f  CERT\_CPP-STR50-g | Avoid overflow due to reading a not zero terminated string Avoid overflow when writing to a buffer Prevent buffer overflows from tainted data Avoid buffer write overflow from tainted data Do not use the 'char' buffer to store input from 'std::cin' |
| PolySpace Bug Finder | R2022b | 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++ Pluhin | 4.10 | S3519 |  |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | [IDS-00-JAV] | Prevent SQL injection. |

| **Noncompliant Code** |
| --- |
| This noncompliant code example shows JDBC code to authenticate a user to a system. The password is passed as a char array, the database connection is created, and then the passwords are hashed. |
| **import** java.sql.Connection;  **import** java.sql.DriverManager;  **import** java.sql.ResultSet;  **import** java.sql.SQLException;  **import** java.sql.Statement;    **class** Login {  **public** Connection getConnection() **throws** SQLException {      DriverManager.registerDriver(**new**              com.microsoft.sqlserver.jdbc.SQLServerDriver());      String dbConnection =        PropertyManager.getProperty("db.connection");      // Can hold some value like      // "jdbc:microsoft:sqlserver://<HOST>:1433,<UID>,<PWD>"  **return** DriverManager.getConnection(dbConnection);    }      String hashPassword(**char**[] password) {      // Create hash of password    }    **public** **void** doPrivilegedAction(String username, **char**[] password)  **throws** SQLException {      Connection connection = getConnection();  **if** (connection == **null**) {        // Handle error      }  **try** {        String pwd = hashPassword(password);          String sqlString = "SELECT \* FROM db\_user WHERE username = '"                           + username +                           "' AND password = '" + pwd + "'";        Statement stmt = connection.createStatement();        ResultSet rs = stmt.executeQuery(sqlString);    **if** (!rs.next()) {  **throw** **new** SecurityException(            "User name or password incorrect"          );        }          // Authenticated; proceed      } **finally** {  **try** {          connection.close();        } **catch** (SQLException x) {          // Forward to handler        }      }    }  } |

| **Compliant Code** |
| --- |
| This compliant solution uses a parametric query with a ? character as a placeholder for the argument. This code also 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):** [Name the principle and explain how it maps to this standard.] |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| The Checker Framework | 2.1.3 | Tainting Checker | Trust and security errors |
| CodeSonar | 7.1p0 | 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 |  | SV.DATA.BOUND  SV.DATA.DB  SV.HTTP\_SPLIT  SV.PATH  SV.PATH.INJ  SV.SQL | Implemented |
| Parasoft Jtest | 2022.1 | CERT.IDS00.TDSQL | Protect against SQL injection |
| SonarQube | 6.7 | 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** | [MEM-50-CPP] | Do not access freed memory. |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, s is dereferenced after it has been deallocated. If this access results in a write-after-free, the vulnerability can be exploited to run arbitrary code with the permissions of the vulnerable process. 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):** [Name the principle and explain how it maps to this standard.] |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 22.10 | dangling\_pointer.use |  |
| Axivion Bauhaus Suite | 7.2.0 | CertC++-MEM50 |  |
| Clang | 3.9 | clang-analyzer-cplusplus.NewDelete | Checked by clang-tidy, but does not catch all violation of this rule |
| CodeSonar | 7.1p0 | 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/write to the target of a free pointer |
| Helix QAC | 2022.3 | C++4303, C++ 4304 |  |
| Klockwork | 2022.3 | 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 | 483S, 484 S | Partially implemented |
| Parasoft C/C++test | 2022.1 | CERT\_CPP-MEM50-a | Do not use resources that have been freed |
| Parasoft Insure++ |  |  | Runtime detection |
| Polyspace Bug Finder | R2022b | CERT C++:MEM50-CPP | Checks for:  Pointer access out of bounds  Deallocation of previously deallocated pointer  Use of previously freed pointer  Rule partially covered. |
| PRQA QA-C++ | 4.4 | 4303, 4304 |  |
| PVS-Studio | 7.22 | V586, V774 |  |
| Splint | 5.0 |  |  |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | [EXP-06-JAV] | Expressions used in assertions must not produce side effects. |

| **Noncompliant Code** |
| --- |
| This noncompliant code attempts to delete all the null names from the list in an assertion. However, the Boolean expression is not evaluated when assertions are disabled. |
| **private** ArrayList<String> names;    **void** process(**int** index) {  **assert** names.remove(**null**); // Side effect    // ...  } |

| **Compliant Code** |
| --- |
| The possibility of side effects in assertions can be avoided by decoupling the Boolean expression from the assertion: |
| **private** ArrayList<String> names;    **void** process(**int** index) {  **boolean** nullsRemoved = names.remove(**null**);  **assert** nullsRemoved; // No side effect    // ...  } |

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

| **Principles(s):** [Name the principle and explain how it maps to this standard.] |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CondeSonar | 7.1p0 | JAVA.STRUCT.SE.ASSERT | Assertion Contains Side Effects (JAVA) |
| PVS-Studio | 7.22 | V6055 |  |
| SonarQube | 6.7 | S3346 | Expressions used in “assert” should not produce side effects |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | [ERR-51-CPP] | Handle all exceptions. |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, neither f() nor main() catch exceptions thrown by throwing\_func(). Because no matching handler can be found for the exception thrown, std::terminate() is called. |
| **void** throwing\_func() noexcept(**false**);    **void** f() {    throwing\_func();  }    **int** main() {    f();  } |

| **Compliant Code** |
| --- |
| In this compliant solution, the main entry point handles all exceptions, which ensures that the stack is unwound up to the main() function and allows for graceful management of external resources. |
| **void** throwing\_func() noexcept(**false**);    **void** f() {    throwing\_func();  }    **int** main() {  **try** {      f();    } **catch** (...) {      // Handle error    }  } |

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

| **Principles(s):** [Name the principle and explain how it maps to this standard.] |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 22.10 | main-function-catch-all-early-catch-all | Partially checked |
| Axivion Bauhaus Suite | 7.2.0 | CertC++-ERR51 |  |
| CodeSonar | 7.1p0 | LANG.STRUCT.UCTCH | Unreadable Catch |
| Helix QAC | 2022.3 | C++4035, C++4036, C++4037 |  |
| Klockwork | 2022.3 | MISRA.CATCH.ALL |  |
| LDRA tool suite | 9.7.1 | 527 S | Partially implemented |
| Parasoft C/C++test | 2022.1 | CERT\_CPP-ERR51-a  CERT\_CPP-ERR51-b | Always catch exceptions Each exception explicitly thrown in the code shall have a handler of a compatible type in all call paths that could lead to that point |
| Polyspace Bug Finder | R2022b | CERT C++:ERR51-CPP | Checks for unhandled exceptions |
| PRQA QA-C++ | 4.4 | 4035, 4036, 4037 |  |
| RuleChecker | 22.10 | main-function-catch-all-early-catch-all | Partially checked |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Input Output | [FIO-50-CPP] | Do not alternately input and output from a file stream without an intervening positioning call. |

| **Noncompliant Code** |
| --- |
| This noncompliant code example appends data to the end of a file and then reads from the same file. However, because 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 >> str;  } |

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

| **Principles(s):** [Name the principle and explain how it maps to this standard.] |
| --- |

**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 | 7.1p0 | IO.IOWOP  IO.OIWOP | Input After Output Without Positioning  Output After Input Without Positioning |
| Helix QAC | 2022.3 | C++4711, C++4712, C++4713 |  |
| Parasoft C/C++test | 2022.1 | CERT\_CPP-FIO50-a | Do not alternately input and output from a stream without an intervening flush or positioning call |
| Polyspace Bug Finder | R2022b | CERT C++:FIO50-CPP | Checks for alternating input and output from a stream without flush or positioning call |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Object Oriented Programming | [OOP-50-CPP] | Do not invoke virtual functions from constructors or destructors. |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, the base class attempts to seize and release an object's resources through calls to virtual functions from the constructor and destructor. However, the B::B() constructor calls B::seize() rather than D::seize(). Likewise, the B::~B() destructor calls B::release() rather than D::release(). |
| **struct** B {    B() { seize(); }  **virtual** ~B() { release(); }    **protected**:  **virtual** **void** seize();  **virtual** **void** release();  };    **struct** D : B {  **virtual** ~D() = **default**;    **protected**:  **void** seize() override {      B::seize();      // Get derived resources...    }    **void** release() override {      // Release derived resources...      B::release();    }  }; |

| **Compliant Code** |
| --- |
| In this compliant solution, the constructors and destructors call a nonvirtual, private member function (suffixed with mine) instead of calling a virtual function. The result is that each class is responsible for seizing and releasing its own resources. |
| **class** B {  **void** seize\_mine();  **void** release\_mine();    **public**:    B() { seize\_mine(); }  **virtual** ~B() { release\_mine(); }    **protected**:  **virtual** **void** seize() { seize\_mine(); }  **virtual** **void** release() { release\_mine(); }  };    **class** D : **public** B {  **void** seize\_mine();  **void** release\_mine();    **public**:    D() { seize\_mine(); }  **virtual** ~D() { release\_mine(); }    **protected**:  **void** seize() override {      B::seize();      seize\_mine();    }    **void** release() override {      release\_mine();      B::release();    }  }; |

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

| **Principles(s):** [Name the principle and explain how it maps to this standard.] |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 22.10 | virtual-call-in-constructor  invlaid\_function\_pointer | Fully checked |
| Axivion Bauhaus Suite | 7.2.0 | CertC++-OOP50` |  |
| Clang | 3.9 | clang-analyzer-alpha.cplusplus.VirtualCall | Checked by clang-tidy |
| CodeSonar | 7.1p0 | LANG.STRUCT.VCALL\_IN\_CTOR  LANG.STRUCT.VCALL\_IN\_DTOR | Virtual Call in Constructor  Virtual Call in Destructor |
| Helix QAC | 2022.3 | C++4260, C++4261, C++4273, C++4274, C++4275, C++4276, C++4277, C++4278, C++4279, C++4280, C++4281, C++4282 |  |
| Klockwork | 2022.3 | CERT.OOP.CTOR.VIRTUAL\_FUNC |  |
| LDRA tool suite | 9.7.1 | 467 S, 92 D | Fully implemented |
| Parasoft C/C++test | 2022.1 | CERT\_CPP-OOP50-a  CERT\_CPP-OOP50-b  CERT\_CPP-OOP50-c  CERT\_CPP-OOP50-d | Avoid calling virtual functions from constructors  Avoid calling virtual functions from destructors  Do not use dynamic type of an object under construction  Do not use dynamic type of an object under destruction |
| Polyspace Bug Finder | R2022b | CERT C++:OOP50-CPP | Checks for virtual function call from constructors and destructors |
| PRQA QA-C++ | 4.4 | 4260, 4261, 4273, 4274,  4275, 4276, 4277, 4278,  4279, 4280, 4281, 4282 |  |
| PVS-Studio | 7.22 | V1053 |  |
| RuleChecker | 22.10 | virtual-call-in-constructor | Fully checked |
| SonarQube C/C++ Plugin | 4.10 | S1699 |  |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Platform Security | [SEC-00-JAV] | Do not allow privileged blocks to leak sensitive information across a trust boundary. |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, the doPrivileged() method is called from the openPasswordFile() method. The openPasswordFile() method is privileged and returns a FileInputStream for the sensitive password file. Because the method is public, it could be invoked by an untrusted caller. |
| **public** **class** PasswordManager {    **public** **static** **void** changePassword() **throws** FileNotFoundException {      FileInputStream fin = openPasswordFile();        // Test old password with password in file contents; change password,      // then close the password file    }    **public** **static** FileInputStream openPasswordFile()  **throws** FileNotFoundException {  **final** String password\_file = "password";      FileInputStream fin = **null**;  **try** {        fin = AccessController.doPrivileged(  **new** PrivilegedExceptionAction<FileInputStream>() {  **public** FileInputStream run() **throws** FileNotFoundException {              // Sensitive action; can't be done outside privileged block              FileInputStream in = **new** FileInputStream(password\_file);  **return** in;            }        });      } **catch** (PrivilegedActionException x) {        Exception cause = x.getException();  **if** (cause **instanceof** FileNotFoundException) {  **throw** (FileNotFoundException) cause;        } **else** {  **throw** **new** Error("Unexpected exception type", cause);        }      }  **return** fin;    }  } |

| **Compliant Code** |
| --- |
| In general, when any method containing a privileged block exposes a field (such as an object reference) beyond its own boundary, it becomes trivial for untrusted callers to exploit the program. This compliant solution mitigates the vulnerability by declaring openPasswordFile() to be private. Consequently, an untrusted caller can call changePassword() but cannot directly invoke the openPasswordFile() method. |
| **public** **class** PasswordManager {  **public** **static** **void** changePassword() **throws** FileNotFoundException {      // ...    }    **private** **static** FileInputStream openPasswordFile()  **throws** FileNotFoundException {      // ...    }  } |

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

| **Principles(s):** [Name the principle and explain how it maps to this standard.] |
| --- |

**Threat Level**

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

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

The first thing we will do in pre-production is assess the threat landscape, changes, and provide a backlog of those changes. We will respond to those threats as needed. We will then design a test-driven practice to help with the security of the software. Next, we can build a secure space for the automation to be executed. To end preproduction, we will verify everything we have implemented and test for vulnerability, functionality, compliance, and security. Full production will take place next. It will start with configuring and deploying security settings. Then we can monitor everything and make sure to log all the data being accessed. We will then respond to threats and block any attacks by turning off necessary services. Finally, we will maintain and stabilize the system.

### 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 |
| --- | --- | --- | --- | --- | --- |
| DLC-50-CPP | High | Probable | Medium | P12 | L1 |
| INT-50-CPP | Medium | Unlikely | Medium | P4 | L3 |
| STR-50-CPP | High | Likely | Medium | P18 | L1 |
| IDS-00-JAV | High | Probable | Medium | P12 | L1 |
| MEM-50-CPP | High | Likely | Medium | P18 | L1 |
| EXP-06-JAV | Low | Unlikely | Low | P3 | L3 |
| ERR-51-CPP | Low | Probable | Medium | P4 | L3 |
| FIO-50-CPP | Low | Likely | Medium | P6` | L2 |
| OOP-50-CPP | Low | Unlikely | Medium | P2 | L3 |
| SEC-00-JAV | Medium | Likely | High | 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 in rest | This policy involves stopping an attacker from gaining access to unencrypted files by ensuring the data is encrypted when on disk. Examples of this are the Payment Card Industry Data Security Standard (PCI) or the Health Insurance Portability and Accountability Act (HIPAA) |
| Encryption at flight | This policy involves stopping an attacker from gaining access to data that is encrypted when it moves over a network. Example of this is Transport Layer Security(TLS) |
| Encryption in use | This policy involves stopping an attacker from gaining access to sensitive data that is being used. That data is never left unsecured regardless of its status. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | This is the first goal in providing a secure system for identifying users who have access to your application. The user needs to be able to authenticate by making sure that the user is who they say they are. Some examples of authentication are things such as security questions and 2 factor authentication. |
| Authorization | This part involves providing the type of access the user will have within the application. This helps to make sure that the right people are accessing certain aspects of the software. An example of this is having role-based access control to help secure the application. Limiting access to sensitive areas helps to make it more secure. |
| Accounting | This part is important in helping to monitor what the user is doing once they have passed through the other parts of the framework. You should know what each user is accessing or attempting to access to help to know if changes need to be made. It can also help to see if the user was authenticated correctly. |

**\***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 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 | 12/08/2022 | Project Milestone | Nick Cleveland |  |
| 3.0 | 12/08/2022 | Finish Project | Nick Cleveland |  |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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