**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 | It is necessary to validate data input from either connecting systems or users in order to ensure the system is both secure and functioning correctly. Proper input validation reduces risks of vulnerabilities such as SQL injection, buffer overflow, and cross-site scripting. Validating input data involves checking the input values against expected patterns, removing potentially malicious segments, and ensuring data input is correctly formatted. |
| 1. Heed Compiler Warnings | A compiler warning is a message generated during the compilation process that identifies potential issues and improper coding practices in the code but doesn’t prevent the code from compiling. In order to ensure that code is secure, developers should address these warnings to avoid bugs and vulnerabilities. |
| 1. Architect and Design for Security Policies | In order to ensure proper security integration within a system, security policies must be integrated into the system design and architecture. Defining security policies early in development is crucial to ensure the system is designed in a secure manner and reduces the need to allocate additional manpower later in development. |
| 1. Keep It Simple | Complex systems may seem better to those unfamiliar with development, but in reality, the more complex a system is, the more vulnerabilities the system has. By keeping the design and development simple, it is easier to develop a system with fewer risks and with greater ease of testing and future updates. |
| 1. Default Deny | The deny by default, or default deny, principle states that access to resources should be restricted by default and only granted when necessary. Doing so reduces the amount of potential avenues for security attacks. |
| 1. Adhere to the Principle of Least Privilege | The principle of least privilege states that a system entity or user should only be granted the minimal level of access that is required for them to perform their required tasks. This principle is put in place to minimize the amount of potential risk by removing the privilege of high-level access for those who do not need it. |
| 1. Sanitize Data Sent to Other Systems | The term “sanitizing” is used in the context of sending data to refer to the act of removing potentially dangerous or harmful content. Doing so reduces the risk of cross-site scripting and SQL injection. |
| 1. Practice Defense in Depth | Defense in depth is the practice of setting up security in layers. By assessing the required amount of protection to properly secure a system balanced with the cost of allocating resources, a layered security implementation can greatly increase security. |
| 1. Use Effective Quality Assurance Techniques | Quality assurance techniques refer to things such as proper testing and code review in order to identify and correct potential issues and develop a code that is not only fully functional but also secure. |
| 1. Adopt a Secure Coding Standard | Adopting a secure coding standard refers to establishing guidelines, to be used across the board, for designing and developing secure code. By setting a standard, individual developers and teams can understand what is and what is not acceptable. |

### 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** |
| --- |
| The modification of p makes the code ill-formed. |
| #include <iostream>    **void** f(**char** c) {  **const** **char** &p = c;    p = 'p'; // Error: read-only variable is not assignable    std::cout << c << std::endl;  } |

| **Compliant Code** |
| --- |
| Removing the const qualifier remedies the issue. |
| #include <iostream>    **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):** 2 – Heed compiler warnings. C++ does not allow the change of values for a reference type. You should not attempt to qualify a reference type, because it will result in undefined behavior. A compiler may and should provide a diagnostic message, however not all compilers will which may result in unexpected behaviors. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion Bauhaus Suite | 7.2.0 | CertC++-DCL52 |  |
| Helix QAC | 2024.4 | C++0014 |  |
| Klockwork | 2024.4 | CERT.DCL.REF\_TYPE.CONST\_OR\_VOLATILE |  |
| Parasoft C/C++ test | 2024.2 | CERT\_CPP-DCL52-a | Never qualify a reference type with ‘const’ or ‘volatile’ |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | [STD-002-CPP] | Do not depend on the order of evaluation for side effects |

| **Noncompliant Code** |
| --- |
| In this example, i is evaluated more than once and is unsequenced making the behavior undefined |
| **void** f(**int** i, **const** **int** \*b) {  **int** a = i + b[++i];    // ...  } |

| **Compliant Code** |
| --- |
| This example is independent of the order of evaluation, resolving the issue. |
| **void** f(**int** i, **const** **int** \*b) {    ++i;  **int** a = i + b[i];    // ...  } |

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

| **Principles(s):** 4—Keep it Simple & 10—Adopt a Secure Coding Standard. In software development, side effects refer to modifying an object, calling a library function, accessing volatile-qualified values, etc. in a way that modifies the execution environment. Using a side effect to obtain a certain outcome is often more complicated and less secure than coding for the expected result. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion Bauhaus Suite | 7.2.0 | CertC++-EXP50 |  |
| Clang | 3.9 | -Wunsequenced | Can detect simple violations of this rule where path-sensitive analysis is not required |
| CodeSonar | 8.3p0 | LANG.STRUCT.SE.DEC  LANG.STRUCT.SE.INC | Side Effects in Expression with Decrement  Side Effects in Expression with Increment |
| Compass/ROSE |  |  | Can detect simple violations of this rule. It needs to examine each expression and make sure that no variable is modified twice in the expression. It also must check that no variable is modified once, then read elsewhere, with the single exception that a variable may appear on both the left and right of an assignment operator |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | [STD-003-CPP] | Guarantee that storage for strings has sufficient space for character data and the null terminator |

| **Noncompliant Code** |
| --- |
| Std::basic\_istream<T>::read() is used to read an unformatted array of 32 characters without guaranteeing the string will be null terminated |
| #include <fstream>  #include <string>    **void** f(std::istream &in) {  **char** buffer[32];  **try** {      in.read(buffer, **sizeof**(buffer));    } **catch** (std::ios\_base::failure &e) {      // Handle error    }      std::string str(buffer);    // ...  } |

| **Compliant Code** |
| --- |
| This solution does not insert a null terminator and instead constructs the std::string object based on the number of characters |
| #include <fstream>  #include <string>    **void** f(std::istream &in) {  **char** buffer[32];  **try** {      in.read(buffer, **sizeof**(buffer));    } **catch** (std::ios\_base::failure &e) {      // Handle error    }    std::string str(buffer, in.gcount());    // ...  } |

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

| **Principles(s):** 3 – Architect and Design for Security Policies & 10 – Adopt a Secure Coding Standard. When you copy a string of data to a buffer that is not large enough to hold that data, the result is a buffer overflow. Buffer overflow is an easy-to-manipulate way to gain unwanted access to a system. Ensuring that proper space is allocated reduces the risk of buffer overflow and leads to a more secure system. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 22.10 | stream-input-char-array | Partially checked + soundly supported |
| CodeSonar | 8.3p0 | MISC.MEM.NTERM  LANG.MEM.BO  LANG.MEM.TO | No space for null terminator  Buffer overrun  Type overrun |
| Helix QAC | 2024.4 | C++5216  DF2835, DF2836, DF2839 |  |
| Klockwork | 2024.4 | NNTS.MIGHT  NNTS.TAINTED  NNTS.MUST  SV.UNBOUND\_STRING\_INPUT.CIN |  |

#### Coding Standard 4

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

| **Noncompliant Code** |
| --- |
| The code modifies the doPrivilegedAction() method to use PreparedStatement instead of java.sql.Statement which still permits an SQL injection |
| **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** |
| --- |
| [Solution uses a parametric query with “?” as a placeholder. The code also validates the username length, preventing an attack via a 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, 7 – Sanitize Data Sent to Other Systems, & 10 – Adopt a Secure Coding Standard. An SQL injection vulnerability arises when elements of an SQL query originate from sources that are untrustworthy. In order to prevent SQL injection, it is essential that input data be validated. Additionally, you must sanitize data sent to other systems. As with most standards, adopting a secure coding standard appropriately applies as that is the first step in secure coding. |
| --- |

**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 | 8.1p0 | JAVA.IO.INJ.SQL | SQL injection |
| Coverity | 7.5 | SQLI  FB.SQL\_PREPARED\_STATEMENT\_GENERATED\_  FB.SQL\_NONCONSTANT\_STRING\_PASSED\_TO\_EXECUTE | Implemented |
| Findbugs | 1.0 | SQL\_NONCONSTANT\_STRING\_PASSED\_TO\_EXECUTE | Implemented |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | [STD-005-CPP] | Properly deallocate dynamically allocated resources |

| **Noncompliant Code** |
| --- |
| Local variable space is passed as the expression to the new operator |
| #include <iostream>    **struct** S {    S() { std::cout << "S::S()" << std::endl; }    ~S() { std::cout << "S::~S()" << std::endl; }  };    **void** f() {    alignas(**struct** S) **char** space[**sizeof**(**struct** S)];    S \*s1 = **new** (&space) S;      // ...    **delete** s1;  } |

| **Compliant Code** |
| --- |
| This solution removes the call to ::operator delete() and calls s1’s destructor |
| #include <iostream>    **struct** S {    S() { std::cout << "S::S()" << std::endl; }    ~S() { std::cout << "S::~S()" << std::endl; }  };    **void** f() {    alignas(**struct** S) **char** space[**sizeof**(**struct** S)];    S \*s1 = **new** (&space) S;      // ...      s1->~S();  } |

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

| **Principles(s):** 3 – Architect and Design for Security Policies & 9 – Use Effective Quality Assurance Techniques. Within a system, memory is a finite resource. In order to prevent unwanted behavior, allocated resources must be deallocated. This, once again, goes back to designing with security in mind as well as using proper quality assurance techniques to ensure the proper deallocation of resources. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 22.10 | Invalid\_dynamic\_memory\_allocation  Dangling\_pointer\_use |  |
| Axivion Bauhaus Suite | 7.2.0 | CertC++-MEM51 |  |
| Clang | 3.9 | Clang-analyzer-cplusplus.NewDeleteLeaks  -Wmismatched-new-delete  Clang-analyzer-unix.MismatchedDeallocator | Checked by clang-tidy, but does not catch all violations of this rule |
| CodeSonar | 8.3p0 | ALLOC.DF  ALLOC.TM  ALLOC.LEAK | Double free  Type mismatch  Leak |

#### 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** |
| --- |
| Code uses the assert() macro to assert a property concerning a memory-mapped structure essential for behavior |
| #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** |
| --- |
| Solution uses a preprocessor conditional statement |
| **struct** timer {    unsigned **char** MODE;    unsigned **int** DATA;    unsigned **int** COUNT;  };    #if (sizeof(struct timer) != (sizeof(unsigned char) + sizeof(unsigned int) + sizeof(unsigned int)))    #error "Structure must not have any padding"  #endif |

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

| **Principles(s):** 9 – Use Effective Quality Assurance Techniques & 10 – Adopt a Secure Coding Standard. Assertions are valuable diagnostic tools when it comes to finding and eliminating software defects that could lead to vulnerabilities. |
| --- |

**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 |
| CodeSonar | 8.3p0 | (customization) | Users can implement a custom check that reports uses of the assert() macro |
| Compase/ROSE |  |  | Could detect violations of this rule merely by looking for calls to assert(), and if it can evaluate the assertion (due to all values being known at compile time), then the code should use static-assert instead; this assumes ROSE can recognize macro invocation |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | [STD-007-CPP] | Handle all exceptions |

| **Noncompliant Code** |
| --- |
| Neither f() nor main() catch exceptions thrown by throwing\_func() |
| [**void** throwing\_func() noexcept(**false**);    **void** f() {    throwing\_func();  }    **int** main() {    f();  } |

| **Compliant Code** |
| --- |
| Solution utilizes main entry point to handle all exceptions |
| **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):** 3 – Architect and Design for Security Policies. When an exception is thrown, control is transferred to the nearest handler with a matching type to the exception. If no matching handler is found, the search extends to surrounding try blocks. This can ultimately lead to a denial-of-service attack. In order to prevent this, the system should be designed to handle all exceptions. |
| --- |

**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 | 8.3p0 | LANG.STRUCT.UCTCH | Unreachable Catch |
| Helix QAC | 2024.4 | C++4035, C++4036, C++4037 |  |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Object Oriented Programming** | [STD-008-CPP] | Do not invoke virtual functions from constructors or destructors |

| **Noncompliant Code** |
| --- |
| Base class attempts to seize and release an object’s resources through a call to virtual functions from constructor and destructor. B::B() calls B::seize() instead of D::seize() and B::~B() call B::release() instead of 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** |
| --- |
| Solution adjusts code so that constructors and destructors call a nonvirtual, private member function instead of a virtual function |
| **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):** 3- Architect and Design for Security Policies & 10 – Adopt a Secure Coding Standard. Because the order of construction starts with base classes and progresses to more derived classes, attempting to call a derived class function from a base class that is under construction can be dangerous because the derived class has not had the opportunity to initialize its resources. Building the system with this in mind will greatly reduce risk. |
| --- |

**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  Invalid-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 | 8.3p0 | LANG.STRUCT.VCALL\_IN\_CTOR  LANG.STRUCT.VCALL\_IN\_DTOR | Virtual Call in Constructor  Virtual Call in Destructor |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Concurrency | [STD-009-CPP] | Ensure actively held locks are released on exceptional conditions |

| **Noncompliant Code** |
| --- |
| Code manipulates shared data and protects the critical section by locking the mutex and unlocking when finished. If an exception occurs while manipulating the data, mutex will not unlock |
| #include <mutex>    **void** manipulate\_shared\_data(std::mutex &pm) {    pm.lock();      // Perform work on shared data.      pm.unlock();  } |

| **Compliant Code** |
| --- |
| Solution catches thrown exceptions when performing work on shared data and unlocks mutex |
| #include <mutex>    **void** manipulate\_shared\_data(std::mutex &pm) {    pm.lock();  **try** {      // Perform work on shared data.    } **catch** (...) {      pm.unlock();  **throw**;    }    pm.unlock(); // in case no exceptions occur  } |

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

| **Principles(s):** 2 – Heed Compiler Warnings & 3 – Architect and Design for Security Policies. Mutexes that are used to protect access to shared data may lock and unlock. If an exception occurs between the lock and unlock and prevents the call of unlock, the mutex will be left in the locked state and no critical sections protected by the mutex can execute. C++ provides lock\_guard, unique\_lock, and shared\_lock that can be initialized with a mutex. In the destructor, it unlocks the mutex should an exception arise. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 8.3p0 | CONCURRENCY.LOCK.NOUNLOCK | Missing Lock Release |
| Helix QAC | 2024.4 | C++5018 |  |
| Parasoft C/C++test | 2024.2 | CERT\_CPP-CON51-a | Do not call lock() directly on a mutex |
| Polyspace Bug Finder | R2024a | CERT C++: CON51-CPP | Checks for lock possible not released on exception (rule fully covered) |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Input Output | [STD-010-CPP] | Close files when they are no longer needed |

| **Noncompliant Code** |
| --- |
| Constructor for std::fstream calls std::basic\_filebuf<T>::open() and std::terminate\_handler called by std::terminate() is std::abort(). Underlying std::basic\_filebuf<T> object not properly 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 solution, std::fstream::close() is called before std::terminate() |
| #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):** 3 – Architect and Design for Security Policies, 4 – Keep it Simple, 10 – Adopt a Secure Coding Standard. As stated previously, resources within a system are finite. In addition, keeping files open that are no longer needed only adds additional layers of vulnerability. Closing programs that are not needed reduces potential risk and frees up resources, reducing vulnerabilities. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 8.3p0 | ALLOC.LEAK | Leak |
| Helix QAC | 2024.4 | DF4786, DF4787, DF4788 |  |
| Klockwork | 2024.4 | RH.LEAK |  |
| Parasoft C/C++test | 2024.2 | CERT\_CPP-FI051-a | Ensure resources are freed |

### 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.

Through the use of automation, we can transition the existing DevOps process into DevSecOps by inserting measures into each step of the DevOps toolchain, as pictured above. Specifically, we insert threat modeling into the “Design” phase. We also incorporate deployment security into the “Build” pase of the toolchain. Within the “Verify and test” phase, we incorporate static application security testing as well as automated security scans.

With the system in the production side of the toolchain, automated testing will continue to ensure the integrity of the sytem is maintained. Additional, a layered formation for security measures, known as “defense in depth” will be in place to provide a higher level of security.

### 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 | Low | P3 | L3 |
| STD-002-CPP | Medium | Probable | Medium | P8 | L2 |
| STD-003-CPP | High | Likely | Medium | P18 | L1 |
| STD-004-JAV | High | Likely | Medium | P18 | L1 |
| STD-005-CPP | High | Likely | Medium | P18 | L1 |
| STD-006-CLG | Low | Likely | High | P1 | L3 |
| STD-007-CPP | Low | Probable | Medium | P4 | L3 |
| STD-008-CPP | Low | Unlikely | Medium | P2 | L3 |
| STD-009-CPP | Low | Probable | Low | P6 | L2 |
| STD-010-CPP | Medium | Unlikely | Medium | P4 | L3 |

### Create Policies for Encryption and Triple A

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

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

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

| 1. **Encryption** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Encryption at rest | Encryption at rest refers to the process of scrambling data for the purpose of only allowing access with a key. This reduces the risk of access by unauthorized individuals. |
| Encryption in flight | Encryption in flight is the process of securing data that is being transmitted over a network. This is essential in order to prevent data leaks and interception. |
| Encryption in use | Encryption in use refers to the process of securing sensitive data even while it is being accessed by a system. This ensures that even if unwanted access is obtained, the receiver will only receive encrypted data. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication refers to the process of verifying the identity of the individual or system attempting to gain access. This is essential in ensuring unwanted access is not obtained. |
| Authorization | Authorization refers to the level of access an individual or system has once authentication has taken place. This ensures that even individuals with access are not able to view or modify data that is not completely necessary to them. |
| Accounting | Accounting refers to the process of keeping a log of activities and events taking place within a system. This provides an accurate detailing of actions taken as a means of identifying potential issues. |

**\***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 |  |
| [Insert text.] | 01/26/2025 | Updated coding standards | Jacob Burchett |  |
| [Insert text.] | 02/16/2025 | Updated standards, added risk assessment, added automated detection/automation, and summary | Jacob Burchett |  |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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