**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 input from all untrusted data sources. This includes command line arguments, network interfaces, environmental variables, and user controlled files. Properly validating inputs can eliminate the vast majority of software vulnerabilities. |
| 1. Heed Compiler Warnings | Compile code using the highest warning level available for your compilers and eliminate warnings by modifying the code. Use static and dynamic analysis tools to detect and eliminate additional security flaws. |
| 1. Architect and Design for Security Policies | Create a software architecture and design your software to implement and enforce security policies. |
| 1. Keep It Simple | Keep the design as simple and small as possible. Complex designs increase the likelihood that errors will be made in their implementation, configuration, and use. Additionally, the effort required to achieve an appropriate level of assurance increases dramatically as security mechanisms become more complex. |
| 1. Default Deny | Base access decisions on permission rather than exclusion. This means that, by default, access in denied and the protection scheme identifies conditions under which access is permitted. |
| 1. Adhere to the Principle of Least Privilege | Every process should execute with the least set of privileges necessary to complete the job. Any elevated permission should only be accessed for the least amount of time required to complete the privileged task. This approach reduces the opportunities an attacker has to execute arbitrary code with elevated privileges. |
| 1. Sanitize Data Sent to Other Systems | Sanitize all data passed to complex subsystems such as command shells, relational databases, and commercial off-the-shelf components. Attackers may be able to invoke unused functionality in these components through the use of SQL, command, or other injection attacks. This is not necessarily an input validation problem because the complex subsystem being invoked does not understand the context in which the call is made. Because the calling process understands the context, it is responsible for sanitizing the data before invoking the subsystem. |
| 1. Practice Defense in Depth | Manage risk with multiple defensive strategies, so that if one layer of defense turns out to be inadequate, another layer of defense can prevent a security flaw from becoming an exploitable vulnerability and/or limit the consequences of a successful exploit. For example, combining secure programming techniques with secure runtime environments should reduce the likelihood that vulnerabilities remaining in the code at deployment time can be exploited in the operational environment. |
| 1. Use Effective Quality Assurance Techniques | Good quality assurance techniques can be effective in identifying and eliminating vulnerabilities. Fuzz testing, penetration testing, and source code audits should all be incorporated as part of an effective quality assurance program. Independent security reviews can lead to more secure systems. External reviewers bring an independent perspective helping to find missed vulnerabilities. |
| 1. Adopt a Secure Coding Standard | Develop and/or apply a secure coding standard for your target development language and platform. |

### 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] | Do not define an unnamed namespace in a header file  https://wiki.sei.cmu.edu/confluence/display/cplusplus/DCL59-CPP.+Do+not+define+an+unnamed+namespace+in+a+header+file |

| **Noncompliant Code** |
| --- |
| The variable “v” is defined in an unnamed namespace within a header file and is accessed from two separate translation units. Each translation unit prints the current value of v and then assigns a new value into it. Since v is defined within an unnamed namespace, each translation unit operates on its own instance of v, giving unexpected output. |
| // a.h  #ifndef A\_HEADER\_FILE  #define A\_HEADER\_FILE    **namespace** {  **int** v;  }    #endif // A\_HEADER\_FILE    // a.cpp  #include "a.h"  #include <iostream>    **void** f() {    std::cout << "f(): " << v << std::endl;    v = 42;    // ...  }    // b.cpp  #include "a.h"  #include <iostream>    **void** g() {    std::cout << "g(): " << v << std::endl;    v = 100;  }    **int** main() {  **extern** **void** f();    f(); // Prints v, sets it to 42    g(); // Prints v, sets it to 100    f();    g();  } |

| **Compliant Code** |
| --- |
| A compliant solution involves defining “v” in only one translation unit but making it externally visible to all translation units, giving the expected behavior. |
| // a.h  #ifndef A\_HEADER\_FILE  #define A\_HEADER\_FILE    **extern** **int** v;    #endif // A\_HEADER\_FILE    // a.cpp  #include "a.h"  #include <iostream>    **int** v; // Definition of global variable v    **void** f() {    std::cout << "f(): " << v << std::endl;    v = 42;    // ...  }    // b.cpp  #include "a.h"  #include <iostream>    **void** g() {    std::cout << "g(): " << v << std::endl;    v = 100;  }    **int** main() {  **extern** **void** f();    f(); // Prints v, sets it to 42    g(); // Prints v, sets it to 100    f(); // Prints v, sets it back to 42    g(); // Prints v, sets it back to 100  } |

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

| **Principles(s):** 3: Architect and Design for Security Policies and 10: Adopt a Secure Coding Standard |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 22.10 | Unnamed-namespace-header | Fully checked |
| Clang | 7.2.0 | cert-dcl59-cpp | Checked by clang-tidy |
| RuleChecker | 22.10 | Unnamed-namespace-header | Fully checked |
| CodeSonar | 8.1p0 | LANG.STRUCT.DECL.ANH | Anonymous Namespace in Header File |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | [STD-002-CPP] | Do not rely on the value of a moved-from object.  https://wiki.sei.cmu.edu/confluence/display/cplusplus/EXP63-CPP.+Do+not+rely+on+the+value+of+a+moved-from+object |

| **Noncompliant Code** |
| --- |
| In this example of noncompliant code the integer values 0-9 are expected to be printed from the std::string rvalue reference to the standard output stream. In this case the object is moved and then reused under the assumption its internal state has been cleared, unexpected output may occur despite not triggering undefined behavior. |
| #include <iostream>  #include <string>    **void** g(std::string v) {    std::cout << v << std::endl;  }    **void** f() {    std::string s;  **for** (unsigned i = 0; i < 10; ++i) {      s.append(1, **static\_cast**<**char**>('0' + i));      g(std::move(s));    }  } |

| **Compliant Code** |
| --- |
| In this compliant example the std::string object is initialized to the expected value on each iteration of the loop. This ensures that the object is in a valid, specified state prior to attempting to access it in “g()”, resulting in the expected output. |
| #include <iostream>  #include <string>    **void** g(std::string v) {    std::cout << v << std::endl;  }    **void** f() {  **for** (unsigned i = 0; i < 10; ++i) {      std::string s(1, **static\_cast**<**char**>('0' + i));      g(std::move(s));    }  } |

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

| **Principles(s):** 1: Validate Input Data, 4: Keep It Simple, and 10: Adopt a Secure Coding Standard |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 8.1p0 | LANG.MEM.NPD.LANG.MEM.UVAR | Null Pointer Dereference Unitialized Variable |
| Helix QAC | 2024.1 | DF4701, DF4702, DF4703 |  |
| Parasoft C/C++test | 2023.1 | CERT\_CPP-EXP63-a | Do not rely on the value of a moved- from object |
| Polyspace Bug Finder | R2023b | Cert C++: EXP63-CPP | Checks for read operations that reads the value of a moved-from object (rule fully covered) |

#### 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.  https://wiki.sei.cmu.edu/confluence/display/cplusplus/STR50-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):** 2: Heed Compiler Warnings and 3:Architect and Design for Security Policies |
| --- |

**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 | 8.1p0 | MISC.MEM.NTERM  LANG.MEM.BO  LANG.MEM.TO | No space for null terminator  Buffer overrun  Type overrun |
| Helix QAC | 2024.1 | C++5216  DF2835, DF2836, DF2839 |  |
| RuleChecker | 22.10 | Stream-input-char-array | Partially checked |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | [STD-004-CPP] | Prevent SQL injection  https://wiki.sei.cmu.edu/confluence/display/java/IDS00-J.+Prevent+SQL+injection |

| **Noncompliant Code** |
| --- |
| In this noncompliant code of an authentication system, the class uses the java.sql.PreparedStatement class properly escapes input strings, preventing SQL injection when used correctly. However, the prepared statement still permits SQL injection attacks by incorporating the unsanitized input argument “username” into the prepared statement. |
| **import** java.sql.Connection;  **import** java.sql.DriverManager;  **import** java.sql.ResultSet;  **import** java.sql.SQLException;  **import** java.sql.Statement;    **class** Login {  **public** Connection getConnection() **throws** SQLException {      DriverManager.registerDriver(**new**              com.microsoft.sqlserver.jdbc.SQLServerDriver());      String dbConnection =        PropertyManager.getProperty("db.connection");      // Can hold some value like      // "jdbc:microsoft:sqlserver://<HOST>:1433,<UID>,<PWD>"  **return** DriverManager.getConnection(dbConnection);    }      String hashPassword(**char**[] password) {      // Create hash of password    }    **public** **void** doPrivilegedAction(      String username, **char**[] password    ) **throws** SQLException {      Connection connection = getConnection();  **if** (connection == **null**) {        // Handle error      }  **try** {        String pwd = hashPassword(password);        String sqlString = "select \* from db\_user where username=" +          username + " and password =" + pwd;        PreparedStatement stmt = connection.prepareStatement(sqlString);          ResultSet rs = stmt.executeQuery();  **if** (!rs.next()) {  **throw** **new** SecurityException("User name or password incorrect");        }          // Authenticated; proceed      } **finally** {  **try** {          connection.close();        } **catch** (SQLException x) {          // Forward to handler        }      }    }  } |

| **Compliant Code** |
| --- |
| Instead this 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):** 1: Validate input data, 7:Sanitize Data Sent to Other Systems, and 10: Adopt a Secure Coding Standard |
| --- |

**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] | Detect and handle memory allocation errors  https://wiki.sei.cmu.edu/confluence/display/cplusplus/MEM52-CPP.+Detect+and+handle+memory+allocation+errors |

| **Noncompliant Code** |
| --- |
| Here we have an array of int is created using ::operator new[] (std::size\_t) and the results of the allocation are not checked. The function is marked as noexcept, so the caller assumes this function does not throw any exceptions. Because ::operator new[] (std::size\_t) can throw an exception if the allocation fails, it could lead to abnormal termination of the program. |
| #include <cstring>    **void** f(**const** **int** \*array, std::**size\_t** size) noexcept {  **int** \*copy = **new** **int**[size];    std::**memcpy**(copy, array, size \* **sizeof**(\*copy));    // ...  **delete** [] copy;  } |

| **Compliant Code** |
| --- |
| In this example we use the std::nothrow, the new operator returns either a null pointer or a pointer to the allocated space. Always test the returned pointer to make sure it is not nullptr before referencing the pointer. This compliant solution handles the error condition appropriately when the returned pointer is nullptr. |
| #include <cstring>  #include <new>    **void** f(**const** **int** \*array, std::**size\_t** size) noexcept {  **int** \*copy = **new** (std::**nothrow**) **int**[size];  **if** (!copy) {      // Handle error  **return**;    }    std::**memcpy**(copy, array, size \* **sizeof**(\*copy));    // ...  **delete** [] copy;  } |

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

| **Principles(s):** 2: Heed Compiler Warnings, 5: Default Deny, and 9: Use Effective Quality Assurance Techniques |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Coverity | 7.5 | CHECKED\_RETURN | Finds inconsistencies in how function call return values are handled |
| Helix QAC | 2024.1 | C++3225, C++3226, C++3227, C++3228, C++3229, C++4632 |  |
| Klocwork | 2024.1 | 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 |  |
| LDRA tool suite | 9.7.1 | 45 D | Partially implemented |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | [STD-006-CPP] | Use a static assertion to test the value of a constant expression  https://wiki.sei.cmu.edu/confluence/display/c/DCL03-C.+Use+a+static+assertion+to+test+the+value+of+a+constant+expression |

| **Noncompliant Code** |
| --- |
| This example of noncompliant code user the assert() macro to assert a property concerning memory-mapped structure that is essential for the code to behave correctly. A runtime assertion is better than nothing, but it needs to be placed in a function and executed otherwise the diagnostic only occurs at runtime and only if the code path containing the assertion is executed. |
| #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** |
| --- |
| Here we use a preprocessor conditional statement this evaluates assertions at compile time and there is no runtime penalty. The use of #error directives allows for clear diagnostic messages for troubleshooting purposes. |
| **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):** 3: Architect and Design for Security Polices, 9: Use Effective Quality Assurance Techniques, and 10: Adopt a Secure Coding Standard |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Clang | 3.9 | Mist-static-assert | Checked by clang-tidy |
| CodeSonar | 8.1p0 | (customization) | Users can implement a custom check that reports uses of the assert() macro |
| ÉCLAIR | 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] | Handle all exceptions |

| **Noncompliant Code** |
| --- |
| In this 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 example, the main entry point handles all exceptions, which ensures that the stack is unwound up to the main() function and allows for clean 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):** 9: Use Effective Quality Assurance Techniques and 10: Adopt a Secure Coding 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 |
| CodeSonar | 8.1p0 | LANG.STRUCT.UCTCH | Unreachable Catch |
| Helix QAC | 2024.1 | C++4035, C++4036, C++4037 |  |
| RuleChecker | 22.10 | main-function-catch-all-early-catch-all | Partially checked |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Object Oriented Programming | [STD-008-CPP] | Copy operations must not mutate the source object  https://wiki.sei.cmu.edu/confluence/display/cplusplus/OOP58-CPP.+Copy+operations+must+not+mutate+the+source+object |

| **Noncompliant Code** |
| --- |
| Here the copy operations for A mutate the source operand by resetting its member variable m to 0. When std::fill() is called, the first element copied will have the original value of obj.m 12, at which point obj.m is set to 0. The subsequent nine copies will all retain the value 0. |
| #include <algorithm>  #include <vector>    **class** A {  **mutable** **int** m;    **public**:    A() : m(0) {}  **explicit** A(**int** m) : m(m) {}      A(**const** A &other) : m(other.m) {      other.m = 0;    }      A& operator=(**const** A &other) {  **if** (&other != **this**) {        m = other.m;        other.m = 0;      }  **return** \***this**;    }    **int** get\_m() **const** { **return** m; }  };    **void** f() {    std::vector<A> v{10};    A obj(12);    std::fill(v.begin(), v.end(), obj);  } |

| **Compliant Code** |
| --- |
| In this compliant example, the copy operations for A no longer mutate the source operand, ensuring that the vector contains equivalent copies of obj. Instead, A has been given more operations that perform the mutation when it is safe to do so. |
| #include <algorithm>  #include <vector>    **class** A {  **int** m;    **public**:    A() : m(0) {}  **explicit** A(**int** m) : m(m) {}      A(**const** A &other) : m(other.m) {}    A(A &&other) : m(other.m) { other.m = 0; }      A& operator=(**const** A &other) {  **if** (&other != **this**) {        m = other.m;      }  **return** \***this**;    }      A& operator=(A &&other) {      m = other.m;      other.m = 0;  **return** \***this**;    }    **int** get\_m() **const** { **return** m; }  };    **void** f() {    std::vector<A> v{10};    A obj(12);    std::fill(v.begin(), v.end(), obj);  } |

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

| **Principles(s):** 9: Use Effective Quality Assurance Techniques and 10: Adopt a Secure Coding Standard |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 8.190 | LANG.FUNCS.COPINC | Copy Operation Parameter is Not const |
| Helix QAC | 2024.1 | C++ 4075 |  |
| Klocwork | 2024.1 | CERT.OOP.COPY\_MUTATES |  |
| Parasoft C/C++test | 2023.1 | CERT\_CPP\_OOP54-a | Copy operations must not mutate the source object. |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Concurrency | [STD-009-CPP] | Avoid deadlock by locking in a predefined order  https://wiki.sei.cmu.edu/confluence/display/cplusplus/CON53-CPP.+Avoid+deadlock+by+locking+in+a+predefined+order |

| **Noncompliant Code** |
| --- |
| This example of noncompliant code depends on the runtime environment and the platform’s scheduler. The program is susceptable to deadlock if thread thr1 attempts to lock ba2’s mutex at the same time thread thr2 attempts to lock ba1’s mutex in the deposit() function. |
| #include <mutex>  #include <thread>    **class** BankAccount {  **int** balance;  **public**:    std::mutex balanceMutex;    BankAccount() = **delete**;  **explicit** BankAccount(**int** initialAmount) : balance(initialAmount) {}  **int** get\_balance() **const** { **return** balance; }  **void** set\_balance(**int** amount) { balance = amount; }  };    **int** deposit(BankAccount \*from, BankAccount \*to, **int** amount) {    std::lock\_guard<std::mutex> from\_lock(from->balanceMutex);      // Not enough balance to transfer.  **if** (from->get\_balance() < amount) {  **return** -1; // Indicate error    }    std::lock\_guard<std::mutex> to\_lock(to->balanceMutex);      from->set\_balance(from->get\_balance() - amount);    to->set\_balance(to->get\_balance() + amount);    **return** 0;  }    **void** f(BankAccount \*ba1, BankAccount \*ba2) {    // Perform the deposits.    std::**thread** thr1(deposit, ba1, ba2, 100);    std::**thread** thr2(deposit, ba2, ba1, 100);    thr1.join();    thr2.join();  } |

| **Compliant Code** |
| --- |
| The compliant code here eliminates the circular wait condition by establishing a predefined order for locking in the deposit() function. Each thread will lick on the basis of the BankAccount ID, which is set when the BankAccount object is initialized. |
| #include <atomic>  #include <mutex>  #include <thread>    **class** BankAccount {  **static** std::atomic<unsigned **int**> globalId;  **const** unsigned **int** id;  **int** balance;  **public**:    std::mutex balanceMutex;    BankAccount() = **delete**;  **explicit** BankAccount(**int** initialAmount) : id(globalId++), balance(initialAmount) {}    unsigned **int** get\_id() **const** { **return** id; }  **int** get\_balance() **const** { **return** balance; }  **void** set\_balance(**int** amount) { balance = amount; }  };    std::atomic<unsigned **int**> BankAccount::globalId(1);    **int** deposit(BankAccount \*from, BankAccount \*to, **int** amount) {    std::mutex \*first;    std::mutex \*second;    **if** (from->get\_id() == to->get\_id()) {  **return** -1; // Indicate error    }      // Ensure proper ordering for locking.  **if** (from->get\_id() < to->get\_id()) {      first = &from->balanceMutex;      second = &to->balanceMutex;    } **else** {      first = &to->balanceMutex;      second = &from->balanceMutex;    }    std::lock\_guard<std::mutex> firstLock(\*first);    std::lock\_guard<std::mutex> secondLock(\*second);      // Check for enough balance to transfer.  **if** (from->get\_balance() >= amount) {      from->set\_balance(from->get\_balance() - amount);      to->set\_balance(to->get\_balance() + amount);  **return** 0;    }  **return** -1;  }    **void** f(BankAccount \*ba1, BankAccount \*ba2) {    // Perform the deposits.    std::**thread** thr1(deposit, ba1, ba2, 100);    std::**thread** thr2(deposit, ba2, ba1, 100);    thr1.join();    thr2.join();  } |

**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, and 10: Adopt a Secure Coding Standard |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 8.1p0 | CONCURRENCY.LOCK.ORDER | Conflicting lock order |
| Coverity | 6.5 | DEADLOCK | Fully implemented |
| Helix QAC | 2024.1 | C++1772, C++1773 |  |
| Parasoft C/C++test | 2023.1 | CERT\_CPP-CON53-a | Do not acquire locks in different order |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Expressions | [STD-010-CPP] | Do not depend on the order of evaluation for side effects  https://wiki.sei.cmu.edu/confluence/display/cplusplus/EXP50-CPP.+Do+not+depend+on+the+order+of+evaluation+for+side+effects |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, I is evaluated more than once in a unsequenced manner, so the behavior of the expression is undefined. |
| **void** f(**int** i, **const** **int** \*b) {  **int** a = i + b[++i];    // ...  } |

| **Compliant Code** |
| --- |
| In this compliant example the example is independent of the order of evaluation of the operands and can be interpreted in only one way. |
| **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):** 2: Heed Compiler Warnings, 4: Keep It Simple, and 10: Adopt a Secure Coding Standard |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Clang | 3.9 | -Wunsequenced | Can detect simple violations of this rule where path-sensitive analysis is not required |
| CodeSonar | 8.1p0 | LANG.STRUCT.SE.DEC  LANG.STRUCT.SE.INC | Side Effects in Expression with Decrement  Side Effects in Expression with Increment |
| ECLAIR | 1.2 | CC2.EXP30 | Fully implemented |
| Helix QAC | 2024.1 | C++3220, C++3221, C++3222, C++3223, C++3228 |  |

### 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.

In order to evolve from DevOps to DevSecOps security must become part of every phase of the process. This starts at the “Assess and Plan” phase where potential threats will be identified and regulations changed to accommodate the security practices necessary to remediate said threats this includes creating a Secure Coding Standard. The “Design” phase is where security practices such as test-driven design can be implemented to add an early layer of security to the system, ensuring the design complies with app best practices such as the OWASP Top Ten is a fairly easy way to mitigate many security vulnerabilities from the start. During the “Build” phase ensuring that any open source libraries used are secure and that all code complies with the Secure Coding Standards created for the project. The “Verify and Test” phase come next and can include static testing and automation to scan the code for any missed vulnerabilities.

From here the Production stage begins starting with the “Transition and Health Check”, this is where penetration testing can be used to find vulnerabilities. Configuration problems with third-party software should be checked at this phase too so ensuring to follow all guidelines from the vendor is crucial. “Monitor and Detect” consists of log collection, event alerting, and intrusion detection this phase is crucial for quick responses to threats which takes place in the next phase “Respond”. Finally “Maintain and Stabilize” where you maintain a security baseline or stabilize your system after an attack back to a stable state and assess against the security baseline.

### Summary of Risk Assessments

Consolidate all risk assessments into one table including both coding and systems standards, ordered by standard number.

| Rule | Severity | Likelihood | Remediation Cost | Priority | Level |
| --- | --- | --- | --- | --- | --- |
| STD-001-CPP | Medium | Unlikely | Medium | Low(4) | 3 |
| STD-002-CPP | Medium | Probable | Medium | Medium(8) | 2 |
| STD-003-CPP | High | Likely | Medium | High(18) | 1 |
| STD-004-CPP | High | Likely | Medium | High(18) | 1 |
| STD-005-CPP | High | Likely | Medium | High(18) | 1 |
| STD-006-CPP | Low | Unlikely | High | Low(1) | 3 |
| STD-007-CPP | Low | Probable | Medium | Low(4) | 3 |
| STD-008-CPP | Low | Likely | Low | Medium(9) | 2 |
| STD-009-CPP | Low | Probable | Medium | Low(4) | 3 |
| STD-010-CPP | Medium | Probable | Medium | Medium(8) | 2 |

### 

### 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 is the practice of encrypting stored data. Encryption at rest protects data that is stored on a device such as hard drive, smartphone, disk, or local storage of an application. Techniques such as application level encryption, database encryption, file system encryption, and full disk encryption can be used to ensure data is protect. |
| Encryption in flight | Encryption in flight is the practice of protecting data as it is travelling between points such as an email being sent from one person to another. This is done by encrypting the data before it is sent and then decrypting said data after it has been delivered. Email encryption, firewalls, and TLS are all tools that can be used to protect data in flight. |
| Encryption in use | Encryption is use is the practice of protecting data that is being actively processed or accessed. The purpose here is to prevent data theft and exposure while being used. The two primary types are Partially homomorphic encryption and Fully homomorphic encryption. Encryption in use can be accomplished using both hardware-based encryption and software-based encryption such as encryption libraries and algorithms. |

Source: <https://phoenixnap.com/blog/encryption-at-rest>

<https://phoenixnap.com/blog/data-in-transit-encryption>

<https://phoenixnap.com/blog/encryption-in-use>

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication is the practice of confirming the identity of a user. This is often accomplished using a username and password. |
| Authorization | Authorization is the practice of specifying the level of access and privileges a user has in a system. Authorization confirms the identity of a user and then authorization enforces the polices of the system. |
| Accounting | Account is the practice of measuring the resources the user accesses while using the system, such as data sent and received. This allows for audits of the system after potential security breachs. |

Source: <https://codebots.com/application-security/aaa-security-an-introduction-to-authentication-authorisation-accounting>

**\***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 | 3/24/2024 | Milestone 3 | James Jones |  |
| 1.2 | 4/13/2024 | Project 1 | James Jones |  |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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