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



# Green Pace Secure Development Policy

## Contents

[Green Pace Secure Development Policy 0](#_Toc172457293)

[Contents 1](#_Toc172457294)

[Overview 3](#_Toc172457295)

[Purpose 3](#_Toc172457296)

[Scope 3](#_Toc172457297)

[Module Three Milestone 3](#_Toc172457298)

[Ten Core Security Principles 3](#_Toc172457299)

[C/C++ Ten Coding Standards 4](#_Toc172457300)

[Coding Standard 1 5](#_Toc172457301)

[Coding Standard 2 7](#_Toc172457302)

[Coding Standard 3 9](#_Toc172457303)

[Coding Standard 4 11](#_Toc172457304)

[Coding Standard 5 14](#_Toc172457305)

[Coding Standard 6 16](#_Toc172457306)

[Coding Standard 7 18](#_Toc172457307)

[Coding Standard 8 20](#_Toc172457308)

[Coding Standard 9 22](#_Toc172457309)

[Coding Standard 10 24](#_Toc172457310)

[Defense-in-Depth Illustration 26](#_Toc172457311)

[Project One 26](#_Toc172457312)

[Revise the C/C++ Standards 26](#_Toc172457313)

[Risk Assessment 26](#_Toc172457314)

[Automated Detection 26](#_Toc172457315)

[Automation 26](#_Toc172457316)

[Summary of Risk Assessments 27](#_Toc172457317)

[Create Policies for Encryption and Triple A 27](#_Toc172457318)

[Map the Principles 28](#_Toc172457319)

[Audit Controls and Management 29](#_Toc172457320)

[Enforcement 29](#_Toc172457321)

[Exceptions Process 29](#_Toc172457322)

[Distribution 30](#_Toc172457323)

[Policy Change Control 30](#_Toc172457324)

[Policy Version History 30](#_Toc172457325)

[Appendix A Lookups 30](#_Toc172457326)

[Approved C/C++ Language Acronyms 30](#_Toc172457327)

[References 30](#_Toc172457328)

## 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 | To eliminate the majority of software vulnerabilities, input should be validated from all untrusted sources. External data sources should be treated as suspicious, which include user-controlled files, command line arguments, environmental variables, as well as network interfaces (Seacord, 2023). |
| 1. Heed Compiler Warnings | Always use the highest warning level available when compiling code. Only eliminate warnings by modifying code meaning don’t suppress warnings unless necessary. To help detect and eliminate vulnerabilities, use static and dynamic analysis tools (Seacord, 2023). |
| 1. Architect and Design for Security Policies | When creating a software architecture, the software should be designed to use security policies and enforce them. For example, consider implementing an appropriate privilege set if and when your system requires different privileges at various times (Seacord, 2023). |
| 1. Keep It Simple | This means keeping the design as small and simple as possible because the bigger the design the higher the likelihood that errors will occur in the design’s implementation, configuration, and use (Seacord, 2023). |
| 1. Default Deny | Instead of basing access decisions on exclusions, they should be based on permissions. Essentially, this means that access should be denied by default and the protection scheme then identifies under what conditions access should be permitted (Seacord, 2023). |
| 1. Adhere to the Principle of Least Privilege | Processes should execute with the least amount of privileges that are needed to complete their job. If elevated permissions are accessed, this should be done in the shortest amount of time possible to complete the privileged task. This reduces the amount of time an attacker is afforded the opportunity to run arbitrary code with these permissions (Seacord, 2023). |
| 1. Sanitize Data Sent to Other Systems | All data that is passed to complex subsystems should be sanitized. These subsystems include command shells, relational databases, and commercial off-the-shelf (COTS) components. With the use of SQL, command, or other means of injection attacks, attackers could potentially invoke unused functionality into these components. This is not necessarily a problem of input validation because the subsystem being invoked does not understand the context in which the call is made, however, the calling process understands it and is therefore responsible for the data to be sanitized before the subsystem is invoked (Seacord, 2023). |
| 1. Practice Defense in Depth | Multiple defense strategies should be utilized to manage risk. Implementing multiple layers of defense helps ensure that if one layer fails another layer is in place to help prevent a security flaw from becoming a vulnerability that can be exploited, which helps to limit the potential consequences of an exploit that is successful. For example, combining two layers, one with secure programming techniques, and another with secure runtime environments, helps to reduce the risk of any vulnerabilities that remain in the code when it is time to be deployed that could be exploited in the operational environment (Seacord, 2023). |
| 1. Use Effective Quality Assurance Techniques | Fuzz testing, penetration testing, and source code audits are all good quality assurance techniques, which when applied can be effective in identifying and eliminating vulnerabilities, and should be incorporated as part of an effective quality assurance program. Independent security reviews, which offer an independent perspective can lead to more secure systems; for example, their perspectives do not include the biases that may exist from internal sources and can help in finding and correcting assumptions that are invalid (Seacord, 2023). |
| 1. Adopt a Secure Coding Standard | A secure coding standard should be developed for and applied to your targeted development language and platform (Seacord, 2023). |

### 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** | DCL-50-CPP | Do not define a C-style variadic function. |

| **Noncompliant Code** |
| --- |
| This example uses a C-style variadic function to add a series of integers together. The function continues to read arguments until a value of 0 is found and therefore calling this function without passing it a value of 0 as an argument (after the first two) results in undefined behavior. Also, if a data type other than an integer is passed to the function, this also results in undefined behavior (Seacord, 2023, Rule 01 Declarations and Initialization). |
| #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** |
| --- |
| For this particular solution, to allow identical behavior for call sites, a variadic function using function parameter pack is used to implement the function add(). Unlike the variadic function used in the non-compliant example, this function does not result in undefined behavior if the list of parameters passed in is not terminated with a 0 value. Also, unlike before, if a value is passed in that is not of the integer data type, this time it does not result in undefined behavior, and the code is ill-formed (Seacord, 2023, Rule 01 Declarations and Initialization). |
| #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** |
| --- | --- | --- | --- | --- |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | EXP-53-CPP | Do not read uninitialized memory. |

| **Noncompliant Code** |
| --- |
| For this non-compliant example, a local variable is uninitialized and used as part of an expression to print the value of it, which results in undefined behavior (Seacord, 2023, Rule 02 Expressions). |
| #include <iostream>    **void** f() {  **int** i;    std::cout << i;  } |

| **Compliant Code** |
| --- |
| This compliant solution shows that the object is initialized before printing its value (Seacord, 2023, Rule 02 Expressions). |
| #include <iostream>    **void** f() {  **int** i = 0;    std::cout << i;  } |

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

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

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

#### 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** |
| --- |
| In this non-compliant code example, the input is unbounded, which leads to a buffer overflow (Seacord, 2023, Rule 05 Characters and Strings). |
| #include <iostream>    **void** f() {  **char** buf[12];    std::cin >> buf;  } |

| **Compliant Code** |
| --- |
| As seen in the following compliant code block, to ensure data is not truncated and to protect against buffer overflows, it is best to use std::string instead of a bounded array (Seacord, 2023, Rule 05 Characters and Strings). |
| #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** |
| --- | --- | --- | --- | --- |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | IDS-00-J | Prevent SQL injection. |

| **Noncompliant Code** |
| --- |
| This non-compliant code example of JDBC code shows that authenticates a user to a system. The password is passed as a character array where the database connection is created, then the passwords are hashed. This code allows a SQL injection attack to occur by using an unsanitized input argument “username” into the SQL command allowing the attacker to use the OR value = value attack, in this case, validuser’ OR ‘1’=’1’ is injected (Seacord, 2023, Rule 00 Input Validation and Data Sanitization). |
| **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 the ‘?’ character as a placeholder for the argument. Also, to prevent an attacker from submitting an arbitrarily long user name, the code validates the length of the username argument (Seacord, 2023, Rule 00 Input Validation and Data Sanitization). |
| **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** |
| --- | --- | --- | --- | --- |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | MEM-50-CPP | Do not access freed memory |

| **Noncompliant Code** |
| --- |
| In this example, s is dereferenced “ s->f();” after it has been deallocated “delete d”. This can result in a write-after-free vulnerability, which can be exploited to run arbitrary code with the permissions that the vulnerable process has (Seacord, 2023, Rule 06 Memory Management). |
| #include <new>    **struct** S {  **void** f();  };    **void** g() noexcept(**false**) {    S \*s = **new** S;    // ...  **delete** s;    // ...    s->f();  } |

| **Compliant Code** |
| --- |
| In this compliant solution, only when the dynamically allocated memory is no longer needed is it then deallocated (Seacord, 2023, Rule 06 Memory Management). |
| #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** |
| --- | --- | --- | --- | --- |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | DCL-03-C | Use a static assertion to test the value of a constant expression |

| **Noncompliant Code** |
| --- |
| In this example, the assert() macro is used to assert a property concerning a memory-mapped structure, which is essential for the code to behave correctly. While this is better than nothing, it should be placed in a function and executed, meaning it would typically be far away from the definition of the actual structure it is referring to (Seacord, 2023, REC 02 Declarations and Initializations). |
| #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** |
| --- |
| A preprocessor conditional statement can be used for assertions involving only constant expressions, which can be seen in the following compliant example. Using #error directives allows for more clearer diagnostic messages. Since assertions are evaluated at compile time, no runtime penalty occurs (Seacord, 2023, REC 02 Declarations and Initializations). |
| **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):** [Name the principle and explain how it maps to this standard.] |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

#### Coding Standard 7

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

| **Noncompliant Code** |
| --- |
| In this particular example of non-compliant code, neither functions f() or main() catch exceptions thrown by throwing\_func(), and since no matching handler can be found for the thrown exception, std::terminate() is called (Seacord, 2023, Rule 08 Exceptions and Error Handling). |
| **void** throwing\_func() noexcept(**false**);    **void** f() {    throwing\_func();  }    **int** main() {    f();  } |

| **Compliant Code** |
| --- |
| As seen in this compliant solution, all exceptions are handled by the main entry point, ensuring that the stack unwound up to the main() function, which allows for graceful management of external resources (Seacord, 2023, Rule 08 Exceptions and Error Handling). |
| **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** |
| --- | --- | --- | --- | --- |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Exceptions | ERR-50-CPP | Do not abruptly terminate the program |

| **Noncompliant Code** |
| --- |
| In this non-compliant code example, f() is registered as an exit handler with std::at\_exit(), which could result in a call to std::terminate because an exception may be thrown from the throwing\_func() (Seacord, 2023, Rule 08 Exceptions and Error Handling). |
| #include <cstdlib>    **void** throwing\_func() noexcept(**false**);    **void** f() { // Not invoked by the program except as an exit handler.    throwing\_func();  }    **int** main() {  **if** (0 != std::**atexit**(f)) {      // Handle error    }    // ...  } |

| **Compliant Code** |
| --- |
| As seen in this compliant code solution, f() function will handle any exceptions thrown by throwing\_func() and therefore will not rethrow (Seacord, 2023, Rule 08 Exceptions and Error handling). |
| #include <cstdlib>    **void** throwing\_func() noexcept(**false**);    **void** f() { // Not invoked by the program except as an exit handler.  **try** {      throwing\_func();    } **catch** (...) {      // Handle error    }  }    **int** main() {  **if** (0 != std::**atexit**(f)) {      // Handle error    }    // ...  } |

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

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

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| String Correctness | STR-51-CPP | Do not attempt to create a std::string from a null pointer |

| **Noncompliant Code** |
| --- |
| For this particular non-compliant example, a std::string object is created as a result of a call to std::getenv(). This call to std::getenv() returns a null pointer on failure, which can lead to undefined behavior when the environmental variable does not exist or when a different error occurs (Seacord, 2023, Rule 05 Characters and Strings). |
| #include <cstdlib>  #include <string>    **void** f() {    std::string tmp(std::**getenv**("TMP"));  **if** (!tmp.empty()) {      // ...    }  } |

| **Compliant Code** |
| --- |
| In this compliant code solution, before the std::string object is constructed, the results from the call to std::getenv() are first checked to see if it is null (Seacord, 2023, Rule 05 Characters and Strings). |
| #include <cstdlib>  #include <string>    **void** f() {  **const** **char** \*tmpPtrVal = std::**getenv**("TMP");    std::string tmp(tmpPtrVal ? tmpPtrVal : "");  **if** (!tmp.empty()) {      // ...    }  } |

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

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

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Memory Protection | MEM-52-CPP | Detect and handle memory allocation errors |

| **Noncompliant Code** |
| --- |
| For this non-compliant code example, an array of type int is created using std::operator new[](std::size\_t), and the results of the allocation are not checked. The caller believes function will not throw any exceptions because it is marked as noexcept. However, because new[](std::size\_t) is able to throw exceptions, this could lead to the program being terminated abnormally (Seacord, 2023, Rule 06 Memory Management). |
| #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 compliant code example, the new operator returns either a null pointer or a pointer to the allocated space when using std::nothrow. This solution handles the error condition appropriately when the returned pointer is nullptr, which should always be tested to ensure it is not nullptr before referencing the pointer (Seacord, 2023, Rule 06 Memory Management). |
| #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):** [Name the principle and explain how it maps to this standard.] |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

### Defense-in-Depth Illustration

This illustration provides a visual representation of the defense-in-depth best practice of layered security.



## Project One

There are seven steps outlined below that align with the elements you will be graded on in the accompanying rubric. When you complete these steps, you will have finished the security policy.

### Revise the C/C++ Standards

You completed one of these tables for each of your standards in the Module Three milestone. In Project One, add revisions to improve the explanation and examples as needed. Add rows to accommodate additional examples of compliant and noncompliant code. Coding standards begin on the security policy.

### Risk Assessment

Complete this section on the coding standards tables. Enter high, medium, or low for each of the headers, then rate it overall using a scale from 1 to 5, 5 being the greatest threat. You will address each of the seven policy standards. Fill in the columns of severity, likelihood, remediation cost, priority, and level using the values provided in the appendix.

### Automated Detection

Complete this section of each table on the coding standards to show the tools that may be used to detect issues. Provide the tool name, version, checker, and description. List one or more tools that can automatically detect this issue and its version number, name of the rule or check (preferably with link), and any relevant comments or description—if any. This table ties to a specific C++ coding standard.

### Automation

Provide a written explanation using the image provided.



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

[Insert your written explanations here.]

### Summary of Risk Assessments

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

| Rule | Severity | Likelihood | Remediation Cost | Priority | Level |
| --- | --- | --- | --- | --- | --- |
| STD-001-CPP | High | Unlikely | Medium | High | 2 |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

### 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 | [Insert text.] |
| Encryption in flight | [Insert text.] |
| Encryption in use | [Insert text.] |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | [Insert text.] |
| Authorization | [Insert text.] |
| Accounting | [Insert text.] |

**\***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 | 07/21/2024 | Updated 10 Core Security Principles, and 10 Coding Standards | Justin Starr |  |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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

## References

Seacord, R. (2023). *Top 10 Secure Coding Practices*. Carnegie Mellon University.

<https://wiki.sei.cmu.edu/confluence/display/seccode/Top+10+Secure+Coding+Practices?focusedCommentId=88044413>