**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 | This principle emphasizes the critical importance of thoroughly validating all input data that a software application receives. Proper input validation helps prevent vulnerabilities like SQL injection or buffer |
| 1. Heed Compiler Warnings | Paying close attention to compiler warnings is crucial for identifying potential code issues and vulnerabilities during the development process. Ignoring these warnings can lead to security weaknesses and system instability. |
| 1. Architect and Design for Security Policies | Security should be considered from the very beginning of the software development lifecycle. This principle urges developers to incorporate security measures into the architectural and design phases, aligning the software with established security policies and requirements. |
| 1. Keep It Simple | Complexity often introduces security risks. Keeping software as simple as possible without sacrificing functionality reduces the attack surface and makes it easier to maintain and secure. |
| 1. Default Deny | In a security context, the default stance should be to deny access unless specifically granted. This principle emphasizes the importance of explicitly defining and implementing access controls, ensuring that unauthorized users or systems are denied access by default. |
| 1. Adhere to the Principle of Least Privilege | Users and systems should have the minimum level of access necessary to perform their tasks. Following this principle helps mitigate the potential impact of security breaches by limiting the privileges of users or processes. |
| 1. Sanitize Data Sent to Other Systems | When sending data to external systems or components, it's crucial to sanitize or validate the data to prevent malicious input from compromising those systems. This practice ensures that the data exchanged does not introduce vulnerabilities. |
| 1. Practice Defense in Depth | This principle encourages multiple layers of security controls and defenses within a system. By implementing various security measures, such as firewalls, intrusion detection, and encryption, organizations can better protect against a wide range of threats. |
| 1. Use Effective Quality Assurance Techniques | Quality assurance extends beyond functionality to include security testing and analysis. Robust quality assurance processes help identify and rectify security vulnerabilities before they can be exploited. |
| 1. Adopt a Secure Coding Standard | Following a well-defined secure coding standard is essential for maintaining a consistent and secure approach to software development. It provides guidelines and best practices that help developers produce code that is less prone to vulnerabilities. |

### C/C++ Ten Coding Standards

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

#### Coding Standard 1

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Type** | STD-001-CPP | Data Type Format-This coding standard defines the format and usage of data types in C/C++ code to ensure consistency and prevent potential security vulnerabilities. |

| **Noncompliant Code** |
| --- |
| The noncompliant code uses inconsistent or inappropriate data types, leading to potential issues and vulnerabilities. In this example, a float data type is used for an integer result, which is incorrect and can result in unexpected behavior. |
| float result = 0; // Inappropriate data type for an integer result |

| **Compliant Code** |
| --- |
| The compliant code uses appropriate and consistent data types, ensuring that the chosen data type aligns with the intended usage. In this corrected example, an int data type is used for an integer result, which is the correct data type for the task. |
| int result = 0; // Correct data type for an integer result |

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

| **Principles(s):** This standard aligns with the "Use Effective Quality Assurance Techniques" principle by ensuring that data types are correctly chosen and consistently applied to prevent issues like type-related vulnerabilities. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | Medium | High | 4 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| ExampleTool | 2.0 | DataTypeCheck | ExampleTool version 2.0 includes a DataTypeCheck for enforcing data type standards. |
| ToolX | 1.5 | TypeValidation | ToolX 1.5 provides a TypeValidation module for automated data type validation. |
| ToolY | 3.1 | DataIntegrity | ToolY 3.1 includes a DataIntegrity checker to ensure data type integrity. |
| ToolZ | 2.5 | TypeInspector | ToolZ 2.5 offers a TypeInspector feature for data type conformity validation. |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | STD-001-CPP | Data Type Format-This coding standard defines the format and usage of data types in C/C++ code to ensure consistency and prevent potential security vulnerabilities. |

| **Noncompliant Code** |
| --- |
| The noncompliant code uses inconsistent or inappropriate data types. |
| float result = 0; // Inappropriate data type for an integer result |

| **Compliant Code** |
| --- |
| The compliant code uses appropriate and consistent data types. |
| int result = 0; // Correct data type for an integer result |

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

| **Principles(s):** This standard aligns with the "Use Effective Quality Assurance Techniques" principle by ensuring that data types are correctly chosen and consistently applied to prevent issues like type-related vulnerabilities. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | Medium | High | 4 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| ExampleTool | 2.0 | DataTypeCheck | ExampleTool version 2.0 includes a DataTypeCheck for enforcing data type standards. |
| ToolX | 1.5 | TypeValidation | ToolX 1.5 provides a TypeValidation module for automated data type validation. |
| ToolY | 3.1 | DataIntegrity | ToolY 3.1 includes a DataIntegrity checker to ensure data type integrity. |
| ToolZ | 2.5 | TypeInspector | ToolZ 2.5 offers a TypeInspector feature for data type conformity validation. |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | STD-002-CPP | String Correctness Standard-This coding standard focuses on ensuring the correctness of string handling and manipulation to prevent common security vulnerabilities. |

| **Noncompliant Code** |
| --- |
| The noncompliant code demonstrates insecure string handling, which can lead to buffer overflows or other security issues. |
| char buffer[20]; strcpy(buffer, userInput); // Vulnerable to buffer overflow |

| **Compliant Code** |
| --- |
| The compliant code uses secure string handling methods to prevent vulnerabilities. In this example, strncpy is used to ensure that the buffer size is not exceeded. |
| char buffer[20]; strncpy(buffer, userInput, sizeof(buffer)); // Safe string 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** |
| --- | --- | --- | --- | --- |
| High | Likely | Medium | High | 4 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| SecureScan | 2.0 | StringCheck | SecureScan 2.0 includes a StringCheck for secure string handling. |
| GuardString | 1.5 | BufferGuard | GuardString 1.5 provides a BufferGuard module for secure string manipulation. |
| SafeString | 3.1 | StringSafety | SafeString 3.1 offers a StringSafety checker for ensuring secure string operations. |
| CodeShield | 2.5 | StringGuard | CodeShield 2.5 has a StringGuard tool for secure string handling. |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | STD-003-CPP | SQL Injection Prevention-This coding standard focuses on preventing SQL injection vulnerabilities by implementing secure coding practices for database interactions. |

| **Noncompliant Code** |
| --- |
| The noncompliant code is susceptible to SQL injection attacks due to improper handling of user input in SQL queries. |
| char query[256];  sprintf(query, "SELECT \* FROM users WHERE username='%s'", userInput); |

| **Compliant Code** |
| --- |
| The compliant code uses parameterized queries to prevent SQL injection vulnerabilities. |
| char query[256]; snprintf(query, sizeof(query), "SELECT \* FROM users WHERE username=?"); |

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

| **Principles(s):** This standard aligns with the "Validate Input Data" principle by emphasizing secure handling of user input in SQL queries to prevent SQL injection attacks. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | Medium | High | 4 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| SQLGuard | 2.0 | SQLInjectionCheck | SQLGuard 2.0 includes an SQLInjectionCheck for detecting SQL injection vulnerabilities. |
| QueryShield | 1.5 | QuerySanitizer | QueryShield 1.5 provides a QuerySanitizer module for secure query parameterization. |
| SafeSQL | 3.1 | SQLSafety | SafeSQL 3.1 offers an SQLSafety checker for secure SQL query construction. |
| CodeShield | 2.5 | SQLGuardian | CodeShield 2.5 has an SQLGuardian tool for SQL injection prevention. |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | STD-004-CPP | Memory Protection Standard-This coding standard focuses on implementing memory protection practices to prevent memory-related vulnerabilities and exploits. |

| **Noncompliant Code** |
| --- |
| The noncompliant code lacks proper memory protection measures, leaving it vulnerable to memory-related vulnerabilities. |
| char buffer[256]; strcpy(buffer, userInput); // Vulnerable to buffer overflow |

| **Compliant Code** |
| --- |
| The compliant code implements memory protection practices to prevent vulnerabilities. In this example, strncpy is used with buffer size checking to prevent buffer overflows. |
| char buffer[256]; strncpy(buffer, userInput, sizeof(buffer)); // Memory-protected string copy |

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

| **Principles(s):** This standard aligns with the "Practice Defense in Depth" principle by emphasizing memory protection as a critical layer of defense against memory-related vulnerabilities. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Medium | Likely | Medium | Medium | 3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| SecureMemory | 2.0 | MemProtection | SecureMemory 2.0 includes a MemProtection checker for memory protection. |
| GuardMem | 1.5 | BufferGuard | GuardMem 1.5 provides a BufferGuard module for secure memory handling. |
| SafeMem | 3.1 | MemorySafety | SafeMem 3.1 offers a MemorySafety checker for secure memory management. |
| CodeShield | 2.5 | MemGuardian | CodeShield 2.5 has a MemGuardian tool for memory protection. |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | STD-005-CPP | Assertions Standard-This coding standard focuses on the use of assertions in code to improve code reliability and security by detecting and handling unexpected conditions. |

| **Noncompliant Code** |
| --- |
| The noncompliant code lacks proper assertions, making it difficult to detect and handle unexpected conditions. |
| int divide(int a, int b) { return a / b; // No assertion to check for b == 0 } |

| **Compliant Code** |
| --- |
| The compliant code uses assertions to check for unexpected conditions and handle them gracefully. |
| int divide(int a, int b) { assert(b != 0); // Assertion to check for b == 0 return a / b; } |

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

| **Principles(s):** This standard aligns with the "Practice Defense in Depth" principle by emphasizing the use of assertions as a defensive measure to catch unexpected conditions and ensure code reliability and security. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Unlikely | Low | Low | 2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| AssertChecker | 2.0 | AssertionCheck | AssertChecker 2.0 includes an AssertionCheck for detecting missing assertions. |
| GuardAssert | 1.5 | AssertGuard | GuardAssert 1.5 provides an AssertGuard module for automated assertion insertion. |
| SafeAssert | 3.1 | AssertionSafety | SafeAssert 3.1 offers an AssertionSafety checker for assertion validation. |
| CodeShield | 2.5 | AssertGuardian | CodeShield 2.5 has an AssertGuardian tool for assertion enforcement. |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | STD-006-CPP | Exceptions Standard-This coding standard focuses on the proper use of exceptions in code to handle exceptional conditions and errors effectively. |

| **Noncompliant Code** |
| --- |
| The noncompliant code lacks proper exception handling, making it difficult to manage exceptional conditions and errors. |
| try { // Code that may throw an exception int result = divide(10, 0); // Division by zero } catch (...) { // Exception handling is missing } |

| **Compliant Code** |
| --- |
| The compliant code uses proper exception handling to manage exceptional conditions and errors gracefully. |
| try { // Code that may throw an exception int result = divide(10, 0); // Division by zero } catch (const std::exception& ex) { // Handle the exception std::cerr << "Exception caught: " << ex.what() << std::endl; } |

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

| **Principles(s):** This standard aligns with the "Practice Defense in Depth" principle by emphasizing proper exception handling as a defense mechanism to gracefully handle errors and exceptional conditions. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Medium | Likely | Low | Medium | 3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| ExceptionScan | 2.0 | ExceptionCheck | ExceptionScan 2.0 includes an ExceptionCheck for detecting missing exception handling. |
| GuardException | 1.5 | ExceptionGuard | GuardException 1.5 provides an ExceptionGuard module for automated exception handling insertion. |
| SafeException | 3.1 | ExceptionSafety | SafeException 3.1 offers an ExceptionSafety checker for exception handling validation. |
| CodeShield | 2.5 | ExceptionGuardian | CodeShield 2.5 has an ExceptionGuardian tool for enforcing proper exception handling. |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Safe File Handling | STD-007-CPP | Safe File Handling Standard-This coding standard focuses on ensuring safe file handling practices in C++ to prevent security vulnerabilities such as file overwrites or unauthorized file access. Safe file handling is crucial to protect sensitive data and prevent security breaches. |

| **Noncompliant Code** |
| --- |
| Noncompliant code lacks proper file handling security measures, making it susceptible to file-related vulnerabilities. |
| std::ofstream file("user\_data.txt"); file << "Sensitive data"; // No error checking or permission checks file.close(); |

| **Compliant Code** |
| --- |
| Compliant code follows safe file handling practices, including error checking and proper permissions handling. |
| std::ofstream file("user\_data.txt", std::ios::app); if (file.is\_open()) { file << "Sensitive data"; // Proper error checking and permission checks file.close(); } else { std::cerr << "Error opening the file." << std::endl; } |

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

| **Principles(s):** This standard aligns with the "Practice Defense in Depth" principle by emphasizing safe file handling as a critical layer of defense against file-related vulnerabilities. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Medium | Likely | Low | Medium | 3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| FileGuard | 2.0 | FileSafetyCheck | FileGuard 2.0 includes a FileSafetyCheck for secure file handling. |
| FileProtector | 1.5 | FilePermissionGuard | FileProtector 1.5 provides a FilePermissionGuard module for permission checks. |
| SafeFileIO | 3.1 | FileAccessSafety | SafeFileIO 3.1 offers a FileAccessSafety checker for safe file I/O. |
| CodeShield | 2.5 | FileGuardian | CodeShield 2.5 has a FileGuardian tool for enforcing safe file handling practices. |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Input Validation | STD-008-CPP | Input Validation Standard-This coding standard focuses on implementing robust input validation practices in C++ to prevent security vulnerabilities related to input data. Proper input validation helps protect against injection attacks, buffer overflows, and other security risks. |

| **Noncompliant Code** |
| --- |
| Noncompliant code lacks adequate input validation, making it vulnerable to input-based attacks and security breaches. |
| std::string username = getRequestParameter("username"); // No input validation; susceptible to SQL injection or other attacks |

| **Compliant Code** |
| --- |
| Compliant code includes thorough input validation to ensure that data is safe and free from potential security threats. |
| std::string username = getRequestParameter("username"); if (isValidUsername(username)) { // Valid username; proceed with safe processing } else { // Invalid input; reject the request or take appropriate action } |

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

| **Principles(s):** This standard aligns with the "Validate Input Data" principle, emphasizing the importance of validating input data to protect against injection attacks and other input-related vulnerabilities. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | Low | High | 4 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| InputGuardian | 2.0 | InputValidator | InputGuardian 2.0 includes an InputValidator for automated input validation. |
| SecureInputCheck | 1.5 | InputGuard | SecureInputCheck 1.5 provides an InputGuard module for secure input handling. |
| SafeInput | 3.1 | InputSafety | SafeInput 3.1 offers an InputSafety checker for input data security. |
| CodeShield | 2.5 | InputGuardian | CodeShield 2.5 has an InputGuardian tool for enforcing input validation practices. |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Secure Password Storage | STD-009-CPP | Secure Password Storage Standard-This coding standard focuses on secure password storage practices in C++ to prevent security vulnerabilities related to storing passwords. Proper password storage is crucial for protecting user accounts and sensitive information. |

| **Noncompliant Code** |
| --- |
| Noncompliant code stores passwords in an insecure manner, making it susceptible to unauthorized access and potential data breaches. |
| std::string userPassword = "myInsecurePassword123";  // Password is stored in plain text, vulnerable to attacks if accessed |

| **Compliant Code** |
| --- |
| Compliant code securely stores passwords using cryptographic hashing and salting to protect them from unauthorized access. |
| std::string userPassword = "mySecurePassword123"; std::string salt = generateRandomSalt(); std::string hashedPassword = hashPassword(userPassword, salt); // Store the hashed password and salt securely |

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

| **Principles(s):** This standard aligns with the "Practice Defense in Depth" and "Adopt a Secure Coding Standard" principles, emphasizing secure password storage as a critical security measure and following established secure coding practices. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | Medium | High | 4 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| PasswordGuard | 2.0 | PasswordSecurity | PasswordGuard 2.0 includes a PasswordSecurity checker for secure password storage. |
| SecurePassword | 1.5 | PasswordProtector | SecurePassword 1.5 provides a PasswordProtector module for password security. |
| HashGuard | 3.1 | HashingSafety | HashGuard 3.1 offers a HashingSafety checker for secure hashing. |
| CodeShield | 2.5 | PasswordGuardian | CodeShield 2.5 has a PasswordGuardian tool for enforcing secure password storage practices. |

### Defense-in-Depth Illustration

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



## Project One

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

### Revise the C/C++ Standards

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

### Risk Assessment

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

### Automated Detection

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

### Automation

Provide a written explanation using the image provided.



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

[Insert your written explanations here.]

### Summary of Risk Assessments

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

| Rule | Severity | Likelihood | Remediation Cost | Priority | Level |
| --- | --- | --- | --- | --- | --- |
| STD-001-CPP | High | Unlikely | Medium | High | 2 |
| STD-002-CPP | Medium | Likely | Low | Medium | 3 |
| STD-003-CPP | Low | Unlikely | High | Low | 1 |
| STD-004-CPP | High | Likely | Medium | High | 4 |
| STD-005-CPP | Medium | Likely | Low | Medium | 3 |
| STD-006-CPP | Low | Unlikely | High | Low | 1 |
| STD-007-CPP | High | Likely | Medium | High | 4 |
| STD-008-CPP | Medium | Likely | Medium | Medium | 3 |
| STD-009-CPP | Low | Unlikely | High | Low | 1 |
| STD-010-CPP | High | Likely | Medium | High | 4 |
| STD-011-CPP | High | Likely | Medium | High | 4 |
| STD-012-CPP | Medium | Likely | Low | Medium | 3 |
| STD-013-CPP | Low | Unlikely | High | Low | 1 |
| STD-014-CPP | Medium | Likely | Medium | Medium | 3 |

### Create Policies for Encryption and Triple A

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

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

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

| 1. **Encryption** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Encryption in rest | Encryption is the process of converting data into a secure and unreadable format using cryptographic techniques. It is used to protect sensitive information from unauthorized access, ensuring confidentiality and data integrity.  Encryption at rest should be applied to all sensitive data stored on company servers, databases, or cloud platforms. This policy ensures that data remains confidential and protected from physical theft or unauthorized access. |
| Encryption at flight | Encryption in flight, also known as data in transit encryption, is the encryption of data while it is being transmitted between two endpoints, such as from a user's device to a server or between servers.  This policy applies to all data transmitted over networks, including internal and external communications. Encryption in flight safeguards data during transmission, preventing eavesdropping and data interception. It is crucial for protecting data privacy and confidentiality during communication. |
| Encryption in use | Encryption in use, or application-level encryption, is the encryption of data while it is actively being processed or used by applications. This protects data even when it is in memory or being manipulated by software.  Encryption in use should be implemented for applications and processes that handle sensitive data. This policy ensures that data remains secure during processing and prevents unauthorized access or data leakage. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication involves verifying the identity of users or devices attempting to access the organization's systems and resources. It ensures that only authorized users gain access. Authentication is essential for controlling access to systems and data. This policy requires all users and devices to undergo proper authentication processes, such as user logins, before granting access. It is crucial to prevent unauthorized access and protect sensitive information. |
| Authorization | Authorization determines what actions or resources an authenticated user or device is permitted to access or modify. It defines user privileges and access levels. Authorization is necessary to restrict access based on roles and responsibilities. This policy ensures that users have the appropriate level of access and permissions to perform their job duties while preventing unauthorized actions. |
| Accounting | Accounting involves monitoring and logging all actions, changes, and access to systems and data. It creates an audit trail for accountability and security analysis. Accounting ensures transparency and accountability for all actions within the organization's systems and data. This policy requires the logging of user logins, changes to the database, addition of new users, user-level access, and files accessed by users. It is crucial for compliance, security, and incident investigation purposes. |

**\***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 |  |
| 2.0 | 09/15/2021 | Updated Security Standards | Jane Doe | Senior Management |
| 3.0 | 10/20/2022 | Added Encryption Requirements | John Smith | Senior Management |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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