**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 | All input data needs to be validated and make sure it adheres to expected formats. This will help prevent injection attacks and other issues like buffer overflows in line with other malicious behavior. Data should be validated if coming from an external source |
| 1. Heed Compiler Warnings | Compiler warnings often signal some potential vulnerabilities or some logic errors. Developers must act on and treat these warnings with serious intent. They need to ensure that code is placed to protect and prevent undefined behavior. |
| 1. Architect and Design for Security Policies | Security should be intergraded from the earliest design phases. Applications must be designed with well-defined security policies in mind such as access control data. |
| 1. Keep It Simple | Simplicity reduces the chance that there can be security flaws. Complex code is hard to test and debug while also taking longer to read through. Developers should also follow minimalistic designs to reduce attack surfaces |
| 1. Default Deny | Systems should deny access by default and only give permissions when explicitly authorized. This makes unintended access and exposure that only validate users so that the process may proceed. |
| 1. Adhere to the Principle of Least Privilege | Processes and users must operate using the minimal set of permissions required to perform the tasks. This makes the impact of potential vulnerabilities or compromised accounts |
| 1. Sanitize Data Sent to Other Systems | Any data that is passed between systems must be sanitized to prevent injection attacks and ensure safe passage. This includes sql html shell and others shared across interfaces |
| 1. Practice Defense in Depth | Multiple layers of security controls should be put in place to protect assets. If one control fails others will be in place to prevent and all out breach |
| 1. Use Effective Quality Assurance Techniques | This will need comprehensive testing, code reviews, static and dynamic while also looking at threat modeling and must be part of the development process to detect and fix vulnerabilities early. |
| 1. Adopt a Secure Coding Standard | Standard secure coding practices reduce inconsistencies and promote code safety using standards like SEI CERT C++ ensures that all developers follow best practices |

### 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 | Ensure correct type conversions Improper type conversions can lead to data truncation. Enforcing strict typing and validating conversions reduce runtime errors and increase program reliability |

| **Noncompliant Code** |
| --- |
| This code converts a large unsigned int value to a signed one causing overflow results |
| unsigned int u = 3000000000U;  int i = u; // Potential data loss |

| **Compliant Code** |
| --- |
| This code checks that the unsigned int value is within the range of a signed int before conversion to prevent undefined behavior |
| unsigned int u = 3000000000U;  int i = 0;  if (u <= static\_cast<unsigned int>(std::numeric\_limits<int>::max())) {  i = static\_cast<int>(u);  } else {  // Handle the error or take alternative action  } |

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

| **Principles(s):** They Enforce what kind of input variable can be accepted and make sure the user input is correct and accessible |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Unlikely | Medium | High | 2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Cppcheck | 2.10 | typeConfusion | Looks for type mismatch, implicit conversions |
| Clang-tidy | 17.0 | cppcoreguidelines | Flag unsafe type narrowing and conversion issues |
| SonarQube | 10.4 | cpp:55144 | Finds unsafe cast, narrowing conversions |
| Fortify SCA | 23.1 | DataFlow analyzer | Identifies unsafe operations on variables caused by incorrect type |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | STD-002-CPP | Validate inputs to functions. Reading or writing data into arrays without validating inputs can cause buffer overflows, leading to crashes or vulnerabilities such as arbitrary code execution |

| **Noncompliant Code** |
| --- |
| This code reads user input into a fixed-size array without validating the index, potentially allowing out-of-bounds access |
| int arr[10];  int index;  std::cin >> index;  arr[index] = 42; // No bounds check |

| **Compliant Code** |
| --- |
| This version includes bounds checking to ensure the index is valid before writing to the array. |
| int arr[10];  int index;  std::cin >> index;  if (index >= 0 && index < 10) {  arr[index] = 42;  } else {  std::cerr << "Error: Index out of bounds.\n";  } |

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

| **Principles(s):** Validate input data to ensure all inputs are with in the accepted ranges. Sanitize data sent to other systems to help eliminate unintended or harmful outputs. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Cppcheck | 2.10 | arrayIndexOutofBounds | Detects array over-indexing, uninitialized values, and improper value usage. |
| Clang-tidy | 17.0 | Clang-analyzer-core.DivideZero | Flag potential divide by zero and logic flaws |
| SonarQube | 10.4 | Cpp:s3519  Cpp:s5789 | Looks at suspicious or insecure uses of values |
| Forify SCA | 23.1 | Tainted value, control flow | Trace tainted values propagation and improper use in control flow. |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | STD-003-CPP | Properly validate string lengths and buffer size. Improper handling of strings especially null-terminated C-style strings can lead to buffer overflows, crashes, or security vulnerabilities |

| **Noncompliant Code** |
| --- |
| This code copies a user string into a fixed-size buffer without checking if the source string is too long. |
| char dest[10];  std::strcpy(dest, "This string is too long"); // Buffer overflow! |

| **Compliant Code** |
| --- |
| This code uses strncpy to limit the number of characters copied, preventing buffer overflow and ensuring the destination is null-terminated. |
| char dest[10];  std::strncpy(dest, "This string is too long", sizeof(dest) - 1);  dest[9] = '\0'; // Ensure null-termination |

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

| **Principles(s):** validate input data -ensures there is no dangerous content or exceeding expected lengths. Then keep it simple using standard and safe string handling functions. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | High | High | 5 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Cppcheck | 2.10 | buffereOverun,strcpy | Flag dangerous C string operations |
| Clang-tidy | 17.0 | Clang-anaylyzer-cplusplus.newdeleteleaks | Detect memory leaks often caused by bad string allocations |
| SonarQube | 10.4 | Cpp:S3659  Cpp:S3518 | Identify unsafe or complex string operations and encoding mismatches |
| Fortify SCA | 23.1 | Buffer Overflow, string manipulation | Detect risks from unsafe string handling |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | STD-004-CPP | Avoid SQL Injection by using parameterized queries. SQL Injection is a major vulnerability that allows attackers to execute arbitrary SQL commands. This can lead to data leakage, unauthorized access, or complete database compromise. |

| **Noncompliant Code** |
| --- |
| This code directly concatenates user input into a SQL statement, making it vulnerable to SQL injection. |
| std::string userInput = getUserInput(); // Imagine this returns "'; DROP TABLE users; --"  std::string query = "SELECT \* FROM users WHERE name = '" + userInput + "';";  db.execute(query); // Dangerous: input is directly inserted |

| **Compliant Code** |
| --- |
| This code uses parameterized queries, which safely bind user input to the SQL command, eliminating the injection risk. |
| std::string userInput = getUserInput();  SQLStatement stmt = db.prepare("SELECT \* FROM users WHERE name = ?");  stmt.bind(1, userInput);  stmt.execute(); // Safe: input is bound, not concatenated |

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

| **Principles(s):** The three are validate input data then default deny and sanitize data sent to other systems. First prevent injection by ensuring user inputs then block unknown or unvalidated queries and commands. Next ensure that user input passed into sql is properly escaped/encoded. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Critical | Likely | High | Critical | 5 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| SonarQube | 10.4 | Cpp:S3649,Cpp:S2077 | Detects Raw sql statements built from user input; recommends prepared statements |
| Forify SCA | 23.1 | SQL injection (dataflow analyzer) | Tracks tainted input to database calls and flags them |
| CodeQl | Latest | Cpp/sql-injection | Queries for dangerous sql construction and use of unsafe user inputs |
| Check marx | 9.x | CX-SEC-65 (SQL Injection Pattern Matcher) | Identifies patterns of unsafe sql usage and verifies protection mechanisms |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | STD-005-CPP | Ensure proper memory management and avoid memory leaks Improper memory handling leads to memory leaks, dangling pointers, and undefined behavior. Leaks degrade performance and may eventually crash the program. Using smart pointers and releasing memory correctly prevents resource exhaustion |

| **Noncompliant Code** |
| --- |
| This code allocates memory with new but does not release it, leading to a memory leak. |
| void processData() {  int\* data = new int[100];  // process data  // memory not freed  } |

| **Compliant Code** |
| --- |
| This code uses std::unique\_ptr to ensure automatic cleanup of dynamically allocated memory. |
| #include <memory>  void processData() {  std::unique\_ptr<int[]> data(new int[100]);  // process data  // memory automatically released when function exits  } |

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

| **Principles(s):** two are keep it simple for simple memory handling and then architect and design for security policies for memory safety. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Cppcheck | 2.10 | Memleak, unintvar | Detects memory leaks, unintilaized variables and use after issues |
| Clang-tidy | [Insert text.] | Clang-anaylyzer-cplusplus.newdeleteleaks | Identify leaks in new/delete usage and mismatched memory |
| valgrind | [Insert text.] | memcheck | Runtime tool to detect memory leaks overflows and use of freed memeory |
| Fortify SCA | 23.1 | Memory leak, pointer dereference | Tracks unsafe dynamic memory behavior including leaks and pointers. |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | STD-006-CPP | Do not rely on assert() for runtime validation. assert() is typically disabled in release builds, so using it for runtime checks can lead to vulnerabilities if invalid inputs go unchecked in production. Critical validations should use explicit error handling instead of assert() to ensure security and stability. |

| **Noncompliant Code** |
| --- |
| This code uses assert() to validate user input, which may be disabled in production builds. |
| #include <cassert>  void setAge(int age) {  assert(age >= 0 && age <= 120); // Not effective in release builds  // continue processing  } |

| **Compliant Code** |
| --- |
| This version explicitly checks the condition and throws an exception if the input is invalid, ensuring the check always runs. |
| #include <stdexcept>  void setAge(int age) {  if (age < 0 || age > 120) {  throw std::invalid\_argument("Age must be between 0 and 120");  }  // continue processing  } |

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

| **Principles(s):** Use assertions for debugging not runtime error handling. These are valuable development time tool used to catch errors by verifying assumptions during testing. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Medium | Likely | Medium | High | 1-3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CPPcheck | 2.13 | useassert | Open source static analyzer that warns dangerous assert() usage |
| Clang-tidy | 18.1 | cert-err33-c (CERT), misc-assert-side-effect | Flag incorrect assertion usage including those with side effects or inputs condtions |
| Fortify SCA | 23.1 | Input validation- use of asseration | Enterprise-grade sast tool that detects misuse of input validation. |
| Coverity static analyzer | 2023.9 | [assert\_side\_effect missing\_check | Commercial sast that flags assertions on external input |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | STD-007-CPP | Only throw exceptions derived from std::exception. Throwing objects not derived from std::exception can lead to inconsistent and unsafe exception handling. Standardizing on std::exception and its derivatives ensures compatibility. |

| **Noncompliant Code** |
| --- |
| This code throws a const char\*, which is not derived from std::exception and does not provide stack trace or type information |
| void readFile(const std::string& filename) {  if (filename.empty()) {  throw "Filename cannot be empty"; // Not derived from std::exception  }  } |

| **Compliant Code** |
| --- |
| This version throws an std::invalid\_argument, providing more structured error information and compatibility with standard exception handling. |
| #include <stdexcept>  void readFile(const std::string& filename) {  if (filename.empty()) {  throw std::invalid\_argument("Filename cannot be empty");  }  } |

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

| **Principles(s):** Use exceptions appropriately and safely. These are to provide a structured way to signal and handle errors and to make sure they are handled correctly. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Clang-tidy | 18.1 | cert-err58-cpp, misc-throw-by-value-catch-by-reference | Detect unhandled exceptions unsafe throw catch usage. |
| Cppcheck | 2.13 | unhandledException, throwInDestructor | Warns on unhandled exceptions, unsafe destructor throws and generic throw abuse |
| Fortify SCA | 23.1 | Rule: "Exception Handling – Missing or Inadequate Catch | Identifies unprotected exceptions paths missing catch blocks and untyped exceptions |
| Covertiy | 2023.9 | UNCAUGHT\_EXCEPT, THROW\_EXCEPTION | Flag mssing exceptions handling, risky rethrows or exceptions in constructors. |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| GOTO statements | STD-008-CPP | Do not use goto statements. The goto statement introduces unstructured control flow, making code harder to understand, maintain, and verify. It can bypass initialization or cleanup code, leading to security vulnerabilities and undefined behavior |

| **Noncompliant Code** |
| --- |
| This code uses goto to jump out of nested logic, which complicates the control flow and risks skipping cleanup steps. |
| #include <iostream>  void process(int n) {  if (n < 0)  goto error;  std::cout << "Processing number: " << n << std::endl;  return;  error:  std::cerr << "Error: negative number" << std::endl;  } |

| **Compliant Code** |
| --- |
| This version uses structured control flow (if statement) and avoids goto, improving readability and correctness. |
| #include <iostream>  void process(int n) {  if (n < 0) {  std::cerr << "Error: negative number" << std::endl;  return;  }  std::cout << "Processing number: " << n << std::endl;  } |

**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** |
| --- | --- | --- |
| Release Resources in All Paths of Execution | STD-009-CPP | Release resources in all paths of execution. Failing to release resources (e.g., memory, file handles, locks) in all execution paths can lead to resource leaks, denial-of-service vulnerabilities, and undefined behavior. Ensuring resources are released regardless of success or failure paths is critical for reliable and secure code |

| **Noncompliant Code** |
| --- |
| This code allocates memory and opens a file but does not properly release them in the event of an error |
| #include <iostream>  #include <fstream>  void loadData(const char\* filename) {  char\* buffer = new char[1024];  std::ifstream file(filename);  if (!file) {  std::cerr << "File not found!" << std::endl;  return; // Memory not released!  }  file.read(buffer, 1024);  delete[] buffer;  } |

| **Compliant Code** |
| --- |
| This version uses RAII principles (std::vector for memory and std::ifstream which closes automatically), ensuring resources are released properly in all execution paths. |
| #include <iostream>  #include <fstream>  #include <vector>  void loadData(const char\* filename) {  std::vector<char> buffer(1024);  std::ifstream file(filename);  if (!file) {  std::cerr << "File not found!" << std::endl;  return;  }  file.read(buffer.data(), buffer.size());  } |

**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** |
| --- | --- | --- |
| [Student Choice] | STD-010-CPP | Ensure Proper Initialization of Variables Before Use. Using uninitialized variables leads to undefined behavior, which can cause unpredictable program results or security vulnerabilities. SEI CERT C++ emphasizes initializing all variables before use to maintain program correctness |

| **Noncompliant Code** |
| --- |
| This code uses a variable without initializing it, potentially resulting in garbage values or program crashes. |
| #include <iostream>  void display() {  int count;  std::cout << "Count is: " << count << std::endl; // count is uninitialized  } |

| **Compliant Code** |
| --- |
| This code initializes the variable count before it is used, ensuring defined behavior. |
| #include <iostream>  void display() {  int count = 0;  std::cout << "Count is: " << count << std::endl;  } |

**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.]

Green Pace can enhance its existing devops pipeline by embedding automated security checks at every stage of the DevSecOps lifestyle. In the pre production phase tools like sonar qube,snyk can enforce secure coding, dependency scanning and infrastructure as code while coding is happening inside of the design build and testing stages. These gates can make it so that only secure code goes through the pipeline.

In production automation continues through the use of these siem tools like splunk for example for real time monitoring, Soar platforms for response and policy as is tools to detect certain drifts. This enforcement of security policies ensures compliance and reduces error and also maintains strong security posture all the way through.

### 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 | High | Likely | High | High | 5 |
| STD-004-CPP | Critical | Likely | High | Critical | 5 |
| STD-005-CPP | High | Unlikely | High | Medium | 4 |
| STD-006-CPP | Medium | Likely | Low | High | 3 |
| STD-007-CPP | High | Likely | Medium | High | 4 |
| STD-008-CPP | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| STD-009-CPP | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| STD-0010-CPP | [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 | **What**: Data encryption at rest protects stored data (on disk or database).  **How**: Use AES-256 to encrypt files/databases on servers or local storage. Encryption keys should be stored separately in a secure key vault.  **Why**: Prevent data theft in case of stolen physical drives or unauthorized server access.  **When**: Always apply to sensitive data like PII, credentials, logs, and financial records stored long-term. |
| Encryption in flight | **What**: Encrypting data as it moves over networks (e.g., internet, APIs).  **How**: Use TLS 1.3 (or TLS 1.2 minimally) for HTTPS, VPN tunnels, and secure sockets.  **Why**: Prevent interception or man-in-the-middle attacks.  **When**: Always required when data is transmitted between systems, users, or services—especially over public networks. |
| Encryption in use | **What**: Protecting data while it is actively being processed in memory.  **How**: Use Trusted Execution Environments (TEEs) like Intel SGX or memory encryption features.  **Why**: Prevent memory scraping, debugging, or attacks on running programs.  **When**: Apply in high-security environments processing sensitive data (e.g., financial transactions, biometric matching). |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | **What**: Verifies the identity of users before granting access.  **How**: Require unique usernames and strong passwords (enforced via policy), multi-factor authentication (MFA), and account lockouts after failed attempts.  **Why**: Prevent unauthorized access through identity impersonation.  **When**: Always required at login; also applies when accessing sensitive resources or executing privileged actions. |
| Authorization | **What**: Determines what a user is allowed to do after authentication.  **How**: Assign role-based access controls (RBAC), limit privileges (least privilege principle), and restrict access to only necessary files or systems.  **Why**: Prevent users from performing unauthorized actions (e.g., modifying critical data or accessing confidential files).  **When**: Every time a user tries to access a resource. |
| Accounting | **What**: Tracks user actions and changes made in the system.  **How**: Use system and application logs to record who accessed what, when, and what changes were made. Store logs securely and monitor with SIEM tools.  **Why**: Helps in auditing, forensics, and detecting suspicious or malicious activity.  **When**: Always—especially during access to sensitive data or system changes. |

**\***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 |  |
| [1.0.] | [6/9/2025 | [Module 3.] | Kyle Serafino | [Insert text.] |
| [1.0.] | [6/16/2025 | [Module 6.] | Kyle Serafino | [Insert text.] |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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