**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 | Ensure all input data in checked for correctness and security to avoid vulnerabilities such as buffer overflows and injection attacks. Validating input early reduces the risk of malicious data entering the system. |
| 1. Heed Compiler Warnings | Always address compiler warning during development to catch potential vulnerabilities and bugs. Ignoring these warnings can lead to undefined behavior and security issues. |
| 1. Architect and Design for Security Policies | Build security measures into the design phase, rather than as an afterthought. Secure design minimizes the surface area for attacks and ensures compliance with organizational standards. |
| 1. Keep It Simple | Avoid overly complex code and systems. Complexity increases the likelihood of security flaws and makes systems harder to maintain and audit. |
| 1. Default Deny | Implement “deny by default” policies to ensure that resources are only accessible to authorized entities. This reduces the chance of unauthorized access. |
| 1. Adhere to the Principle of Least Privilege | Restrict users and processes to the minimum permissions needed to perform their tasks. This limits the potential impact of a compromised account or process. |
| 1. Sanitize Data Sent to Other Systems | Always clean and validate data before sending it to other systems to prevent injection attacks and ensure data integrity. |
| 1. Practice Defense in Depth | Use multiple layers of security controls to protect against various threats. For example, combining firewalls, input validations, and encryption provides a stronger defense. |
| 1. Use Effective Quality Assurance Techniques | Implement automated testing, code reviews, and static analysis tools to identify vulnerabilities early in the development process. |
| 1. Adopt a Secure Coding Standard | Follow a well-defined coding standard, such as the SEI CERT C++ Coding Standard, to ensure consistency and security across all codebases. |

### 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 Validation |

| **Noncompliant Code** |
| --- |
| Incorrect use of a fixed-size array that causes buffer overflow. |
| char buffer[10];  strcpy(buffer, “This string is too long!”); // Buffer overflow |

| **Compliant Code** |
| --- |
| User safer alternatives like strncpy to prevent overflow. |
| char buffer[10];  strncpy(buffer, “Safe string”, sizeof(buffer) – 1); // Safe usage  buffer[sizeof(buffer) – 1] = ‘\0’; // Ensure null termination |

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

| **Principles(s):** Validate Input Data, Adopt a Secure Coding Standard |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| SonarQube | 9.3 | CERT-DCL03-C | Identifies unsafe data type usage and buffer overflows. |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | STD-002-CPP | Data Range Validation |

| **Noncompliant Code** |
| --- |
| Accepting user-provided data without range validation |
| int age = get\_user\_input(); // No range check |

| **Compliant Code** |
| --- |
| Validate user input against an acceptable range. |
| int age = get\_user\_input();  if (age < 0 || age > 150) {  throw std::out\_of\_range(“Invalid age”);  } |

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

| **Principles(s):** Validate Input Data, Architect and Design for Security Policies. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Fortify SCA | 20.2 | Range Validation Check | Flags missing range validation in user input. |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | STD-003-CPP | String Handling |

| **Noncompliant Code** |
| --- |
| Using unsafe functions like strcpy. |
| char buffer[50];  strcpy(buffer, “This could overflow!”); // Unsafe |

| **Compliant Code** |
| --- |
| Use modern, safer alternatives like std::string |
| std::string buffer = “This is safe!”; |

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

| **Principles(s):** Validate Input Data, Keep It Simple. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Clang Static Analyzer | 14.0 | String Safety Analyzer | Detects unsafe string operations. |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | STD-004-CPP | Parameterized Queries |

| **Noncompliant Code** |
| --- |
| Concatenating user input directly into an SQL query. |
| std::string query = “SELECT \* FROM users WHERE name = ‘” + username + “’”; |

| **Compliant Code** |
| --- |
| Use parameterized queries with placeholders |
| sql::PreparedStatement \*stmt;  stmt = conn->prepareStatement(“SELECT \* FROM users WHERE name = ?”);  stmt->setString(1, username); |

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

| **Principles(s):** Sanitize Data Sent to Others Systems, Practice Defense in Depth. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| OWASP ZAP | 2.10.0 | SQL Injection Test Module | Simulates SQL Injection attacks to identify vulnerabilities. |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | STD-005-CPP | Avoiding memory leaks and unsafe memory usage reduces vulnerabilities such as use-after-free and dangling pointers. |

| **Noncompliant Code** |
| --- |
| Failing to free dynamically allocated memory. |
| int\* ptr = new int[10]; // Memory leak: no delete statement |

| **Compliant Code** |
| --- |
| Ensure all allocated memory is properly deallocated. |
| int\* ptr = new int[10];  delete[] ptr; // Memory is freed |

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

| **Principles(s):** Heed Compiler Warnings, Practice Defense in Depth |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Valgrind | 3.17.0 | Memory Leak Detection Tool | Detects unfreed memory and danglin pointers. |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | STD-006-CPP | Assertions should only be used during development to catch bugs, they should not replace runtime error handling in production. |

| **Noncompliant Code** |
| --- |
| Using assertions for error handling in production. |
| assert(file != nullptr); // Unsafe for production |

| **Compliant Code** |
| --- |
| Use assertions in development and proper error handling in production. |
| if (file == nullptr) {  throw std::runtime\_error(“File not found!”);  } |

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

| **Principles(s):** Use Effective Quality Assurance Techniques, Adhere to the Principle of Least Privilege. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| SonarLint | 6.9 | Assertion Use Analysis Module | Identifies unsafe use of assertions in production. |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | STD-007-CPP | Exceptions allow for better error handling, as they separate error reporting from normal control flow. |

| **Noncompliant Code** |
| --- |
| Using return codes for error handling. |
| int openFile(const char\* filename) {  if (!fileExists(filename)) return -1;  return 0;  } |

| **Compliant Code** |
| --- |
| Use exceptions to handle errors instead of return codes. |
| void openFile(const char\* filename) {  if (!fileExists(filename)) {  throw std::runtime\_error(“File does not exist”);  }  } |

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

| **Principles(s):** Keep It Simple, Adopt a Secure Coding Standard. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Clang Static Analyzer | 14.0 | Exception Handling Analyzer | Identifies improper use of return codes instead of exceptions. |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Error Logging | STD-008-CPP | Logs should not expose sensitive data to unauthorized users. |

| **Noncompliant Code** |
| --- |
| Logging sensitive information in plaintext. |
| log(“User password: “ + password); // Unsafe |

| **Compliant Code** |
| --- |
| Mask sensitive data before logging. |
| log(“User password: [REDACTED]”); |

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

| **Principles(s):** Default Deny, Practice Defense in Depth |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Splunk | 8.2 | Sensitive Data Logging Module | Flags sensitive data exposed in log files. |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Concurrency Control | STD-009-CPP | Improper handling of shared resources can lead to race conditions and unpredictable behavior. |

| **Noncompliant Code** |
| --- |
| Accessing a shared resource without locking. |
| int counter = 0;  void increment() {  counter++; // Unsafe: no synchronization |

| **Compliant Code** |
| --- |
| Use locks to synchronize access to shared resources. |
| std::mutex mtx;  int counter = 0;  void increment() {  std::lock\_guard<std::mutex> lock(mtx);  counter++; // Safe: synchronize access  } |

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

| **Principles(s):** Practice Defense in Depth, Architect and Design for Security Policies. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Coverity | 2024.3 | Thread Safety Analysis Module | Detects race conditions and improper synchronization in multithreaded code. |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Cryptographic Practices | STD-010-CPP | Avoid implementing custom cryptographic algorithms, use proven libraries to ensure security. |

| **Noncompliant Code** |
| --- |
| Using a custom encryption algorithm. |
| void customEncrypt(std::string& data) {  for (auto& c : data) c ^= 0x5A; // Weak and insecure encryption |

| **Compliant Code** |
| --- |
| Use a secure library such as OpenSSL for encryption. |
| void encryptData(const std::string& data) {  EVP\_CIPHER\_CTX\* ctx = EVP\_CIPHER\_CTX\_new();  // Uses AES encryption with OpenSSL  } |

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

| **Principles(s):** Use Effective Quality Assurance Techniques, Adopt a Secure Coding Standard. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| OpenSSL | 3.0 | Encryption Library Usage Analysis | Ensures secure cryptographic practices and flags custom encryption implementations. |

### 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 | Likely | Medium | High | 4 |
| STD-002-CPP | Medium | Likely | Low | Medium | 3 |
| STD-003-CPP | High | Likely | Low | High | 3 |
| STD-004-CPP | Critical | High | Medium | High | 5 |
| STD-005-CPP | High | Medium | High | High | 4 |
| STD-006-CPP | Medium | Low | Low | Low | 2 |
| STD-007-CPP | Medium | Medium | Low | Medium | 3 |
| STD-008-CPP | High | Medium | Low | High | 4 |
| STD-009-CPP | High | High | Medium | High | 5 |
| STD-010-CPP | Critical | Low | High | High | 5 |

### 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 | * **Policy:** Use AES-256 for encrypting databases, file systems, and backups to protect against physical theft or unauthorized access. * **When to Apply:** Applied during data storage |
| Encryption in flight | * **Policy:** Use TLS 1.3 for secure data transmissions over networks to prevent eavesdropping or data interception. * **When to Apply:** Applied to all APIs, websites, and communication protocols. |
| Encryption in use | * **Policy:** Use memory-level encryption or secure enclaves for sensitive data processing to prevent exposure during execution. * **When to Apply:** Applied in volatile memory operations involving sensitive data. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | * **Policy:** Implement multi-factor authentication (MFA) and password hashing using bcrypt. * **When to Apply:** Applied during user login and access requests. |
| Authorization | * **Policy:** Enforce role-based access control (RBAC) with least privilege principles. * **When to Apply:** Applied to define access levels for all users and systems. |
| Accounting | * **Policy:** Log all user actions, changes to the database, and file access for compliance auditing. * **When to Apply:** Continuous logging for user actions and 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 |  |
| 1.1 | 11/26/2024 | Revised to include coding standards 1-6 and initial encryption and Triple-A framework policies. | Joshua Perry | [Insert text.] |
| 1.2 | 12/07/2024 | Completed risk assessments, detailed automation detection strategies, and finalized encryption and Triple-A framework policies. | Joshua Perry | [Insert text.] |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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