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



# Green Pace Secure Development Policy

# Brad Mills

## 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 | Always validate input data to ensure it meets the expected format, type, and length. This prevents attackers from exploiting vulnerabilities like buffer overflows, SQL injection, and cross-site scripting. For example, ensuring a string input does not exceed a certain length or validating that numeric data falls within an expected range can mitigate potential threats at the entry point. |
| 1. Heed Compiler Warnings | Compiler warnings often indicate potential vulnerabilities or unsafe coding practices. Ignoring these warnings can lead to unintended behaviors or exploitable flaws in the application. Developers should treat warnings seriously, using them as an opportunity to address potential issues early in the development cycle before they escalate into critical security risks. |
| 1. Architect and Design for Security Policies | Security should be a fundamental part of system design and architecture. This includes strategies such as separating sensitive data from less secure environments and enforcing security policies like access controls and encryption. By building systems with security in mind, developers can reduce the attack surface and make it harder for vulnerabilities to be exploited. |
| 1. Keep It Simple | Complexity is the enemy of security. Overly complicated code or system designs increase the risk of errors and make vulnerabilities harder to detect and fix. Developers should aim to keep their codebase clean and maintainable, focusing on simplicity and clarity to minimize potential security flaws. |
| 1. Default Deny | By default, systems should deny access to all resources unless explicitly allowed. This principle ensures that accidental or unauthorized access is prevented. For example, a firewall should block all incoming traffic unless a specific rule allows it. This reduces the risk of leaving sensitive areas unintentionally exposed. |
| 1. Adhere to the Principle of Least Privilege | Every user, process, or system component should have only the permissions necessary to perform its tasks—nothing more. This limits the damage that can occur if an account or process is compromised. For instance, a user responsible for updating content should not have access to system-level configurations. |
| 1. Sanitize Data Sent to Other Systems | Data sent to external systems must be sanitized to ensure it is safe and complies with expected formats. This helps prevent injection attacks, such as SQL injection or cross-site scripting, that could harm other systems. Proper sanitization protects both the sending and receiving systems from potential threats. |
| 1. Practice Defense in Depth | Employing multiple layers of security ensures redundancy and improves resilience. If one layer of defense fails, others are still in place to protect the system. For example, combining strong authentication, encryption, and intrusion detection creates a more robust security posture than relying on a single mechanism. |
| 1. Use Effective Quality Assurance Techniques | Regular testing, secure coding reviews, and static analysis tools can catch vulnerabilities early in the development process. Ensuring quality throughout the software lifecycle reduces the chance of exploitable flaws reaching production. Comprehensive testing also builds confidence in the system’s reliability and security. |
| 1. Adopt a Secure Coding Standard | Following an established secure coding standard, such as the SEI CERT C++ Coding Standard, ensures consistent implementation of best practices across the team. These standards address common vulnerabilities and provide developers with guidelines to write secure and maintainable code. |

### 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 | Using appropriate data types reduces memory waste and prevents unexpected behaviors due to type mismatches. Choosing the correct type ensures data consistency and protects against potential overflows or truncation. |

| **Noncompliant Code** |
| --- |
| This code uses an inappropriate data type that could lead to unexpected behavior or overflows. |
| unsigned int age = -5; // Invalid assignment |

| **Compliant Code** |
| --- |
| This code uses the appropriate data type and ensures values stay within valid ranges. |
| unsigned int age = 25; // Correct usage |

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

| **Principles(s):** Validate Input Data - Ensures that values are appropriate for their data type. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Clang-tidy | 14.0 | Misc-inaccurate-data-type | Detects inappropriate/unsafe data types |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | STD-002-CPP | Ensuring data values are validated reduces the risk of buffer overflows, underflows, and other vulnerabilities caused by improper input or calculations. |

| **Noncompliant Code** |
| --- |
| This code does not validate input, leading to potential issues. |
| int input = atoi(userInput); // Unsafe conversion |

| **Compliant Code** |
| --- |
| This code uses validation to ensure input values are within the expected range. |
| int input = std::stoi(userInput);  if (input < 0 || input > 100) {  throw std::out\_of\_range("Input out of range");  } |

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

| **Principles(s):** Sanitize Data Sent to Other Systems - Ensures only valid data is processed. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| SonarQube | 9.0 | S5758 | Flags unsafe input conversions |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | STD-003-CPP | Improper string handling can lead to buffer overflows and security vulnerabilities, especially when dealing with user input or external data. |

| **Noncompliant Code** |
| --- |
| Using unbounded string operations introduces risks of buffer overflow. |
| char buffer[10];  strcpy(buffer, userInput); // Unsafe |

| **Compliant Code** |
| --- |
| Use safer alternatives to prevent buffer overflows. |
| char buffer[10];  strncpy(buffer, userInput, sizeof(buffer) - 1); // Safe usage  buffer[sizeof(buffer) - 1] = '\0'; // Null-terminate |

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

| **Principles(s):** Practice Defense in Depth - Ensures that multiple safeguards are in place for secure string operations. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Cppcheck | 2.6 | Buffer-overrun | Identifies risky buffer manipulations |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | STD-004-CPP | Sanitizing input before using it in SQL queries is critical to prevent SQL injection attacks. Unsafe queries can allow attackers to manipulate databases and access sensitive information. |

| **Noncompliant Code** |
| --- |
| Directly concatenating user input into an SQL query can lead to SQL injection vulnerabilities. |
| std::string query = "SELECT \* FROM users WHERE username = '" + username + "';";  db.execute(query); |

| **Compliant Code** |
| --- |
| Use parameterized queries to prevent SQL injection by separating code from data. |
| std::string query = "SELECT \* FROM users WHERE username = ?";  db.prepare(query);  db.bind(1, username);  db.execute(); |

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

| **Principles(s):** Sanitize Data Sent to Other Systems - Ensures that untrusted data is sanitized before use. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| SonarQube | 9 | S3649 | Detects dynamic SQL queries and flags bad practices |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | STD-005-CPP | Proper memory management prevents issues like buffer overflows, dangling pointers, and memory leaks, which can lead to unpredictable behavior and security vulnerabilities. |

| **Noncompliant Code** |
| --- |
| Failing to release dynamically allocated memory can lead to memory leaks. |
| int\* array = new int[10];  // Forgot to delete array |

| **Compliant Code** |
| --- |
| Use smart pointers to ensure memory is automatically released when no longer needed. |
| std::unique\_ptr<int[]> array(new int[10]); |

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

| **Principles(s):** Adhere to the Principle of Least Privilege - Ensures that resources are managed securely. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Valgrind | 3.18 | Memcheck | Detects memory leaks improper memory access |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | STD-006-CPP | Assertions validate assumptions made by the program during runtime and help identify bugs early. However, they should not be used for input validation or production error handling. |

| **Noncompliant Code** |
| --- |
| Using assertions for input validation is inappropriate and can lead to unexpected program termination in production. |
| assert(input > 0); // Unsafe for production |

| **Compliant Code** |
| --- |
| Use assertions for debugging and runtime checks, not for user input validation. |
| if (input <= 0) {  throw std::invalid\_argument("Input must be greater than 0");  } |

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

| **Principles(s):** Use Effective Quality Assurance Techniques - Ensures that assumptions are tested during development. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Clang-tidy | 14 | Misc-assert | Flags misuse of assertions |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | STD-007-CPP | Proper use of exceptions ensures that errors are handled consistently and do not leave the application in an undefined state. Avoid throwing exceptions for control flow. |

| **Noncompliant Code** |
| --- |
| Throwing exceptions for control flow creates unnecessary overhead and can complicate debugging. |
| try {  throw std::runtime\_error("Error");  } catch (...) {  // Catch-all handler  } |

| **Compliant Code** |
| --- |
| Use exceptions for error handling, not as a substitute for normal control flow. |
| if (!processData()) {  throw std::runtime\_error("Processing failed");  } |

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

| **Principles(s):** Keep It Simple - Ensures error handling is clear and predictable. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Cppcheck | 2.6 | Exception-usage | Flags inappropriate exceptions |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Input Validation | STD-008-CPP | Input validation ensures that all user-provided data is checked for correctness and adherence to expected formats before processing, reducing vulnerabilities like injection attacks or crashes. |

| **Noncompliant Code** |
| --- |
| Accepting user input without validation can lead to unexpected behavior or security issues. |
| std::string username;  std::cin >> username; // No validation |

| **Compliant Code** |
| --- |
| Validate input to ensure it meets expected criteria before processing. |
| std::string username;  std::cin >> username;  if (username.empty() || username.length() > 20) {  throw std::invalid\_argument("Invalid username length");  } |

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

| **Principles(s):** Validate Input Data - Ensures that all inputs are checked for correctness before use. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| SonarQube | 9 | S4823 | Flags potential input validation issue(s) |
| [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** |
| --- | --- | --- |
| Resource Management | STD-009-CPP | Properly managing system resources (e.g., file handles, network connections) ensures that resources are released when no longer needed, preventing resource leaks and availability issues. |

| **Noncompliant Code** |
| --- |
| Failing to release a file handle can lead to resource exhaustion. |
| std::ofstream file("data.txt");  // No file.close() call |

| **Compliant Code** |
| --- |
| Use RAII (Resource Acquisition Is Initialization) to manage resources automatically. |
| {  std::ofstream file("data.txt");  file << "Sample data";  } // File is closed automatically when out of scope |

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

| **Principles(s):** Practice Defense in Depth - Ensures resources are managed securely and consistently. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Valgrind | 3.18 | ResourceLeak | Detects resource leaks and bad management |
| [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** |
| --- | --- | --- |
| Secure Logging | STD-010-CPP | Logs should not contain sensitive information such as passwords, private keys, or personal data. This prevents accidental exposure of sensitive information in logs. |

| **Noncompliant Code** |
| --- |
| Logging sensitive information exposes it to unauthorized access. |
| std::cout << "User password: " << password << std::endl; |

| **Compliant Code** |
| --- |
| Log minimal information and redact sensitive details where necessary. |
| std::cout << "User logged in successfully" << std::endl; |

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

| **Principles(s):** Adopt a Secure Coding Standard - Ensures logging practices adhere to security best practices. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Cppcheck | 2.6 | Secure-logging | Flags logging practices that expose sensitive info |
| [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.

Automation is key to ensuring consistent enforcement of coding standards and policy compliance across the DevSecOps pipeline. At Green Pace, automation should be integrated at the following stages:

Plan: Use tools like SonarQube to analyze existing codebases for vulnerabilities during planning and design.

Verify: Perform static application security testing (SAST) and dynamic application security testing (DAST) using tools like Checkmarx or Fortify. These tools detect vulnerabilities in code and runtime environments.

Release: Automate software signing to verify integrity before deployment using GPG or similar tools.

Detect: Use runtime application self-protection (RASP) tools to monitor for unusual behavior or threats in production environments.

Respond: Employ security orchestration tools to automate responses to detected threats, such as blocking IPs or patching vulnerabilities.

### 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 | High | Medium | Medium | High | 5 |
| STD-003-CPP | High | High | Medium | High | 5 |
| STD-004-CPP | Critical | High | Medium | Very High | 5 |
| STD-005-CPP | High | Medium | Medium | High | 4 |
| STD-006-CPP | Medium | Low | Low | Medium | 3 |
| STD-007-CPP | Medium | Medium | Low | Medium | 3 |
| STD-008-CPP | High | Medium | Low | High | 4 |
| STD-009-CPP | High | Medium | Medium | High | 4 |
| STD-010-CPP | Medium | Medium | Low | 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 at rest | What it is: Encrypting data stored on a physical medium, such as a disk or database, to prevent unauthorized access if the storage device is compromised.  How it’s used: Use AES-256 encryption for databases, file systems, and backups. Securely manage encryption keys using a hardware security module (HSM) to prevent key leakage.  Why it applies: Protecting data at rest is critical to ensure confidentiality and compliance with regulatory standards. |
| Encryption in flight | What it is: Encrypting data while it is being transmitted over a network, ensuring secure communication between endpoints.  How it’s used: Implement TLS 1.3 for secure web traffic and encrypted communication protocols like HTTPS and VPNs.  Why it applies: Protects data from being intercepted by attackers during transit. |
| Encryption in use | What it is: Encrypting data while it is actively being processed in volatile memory.  How it’s used: Leverage technologies like Intel SGX or AMD SEV to secure memory regions used during computation.  Why it applies: Prevents attackers from accessing sensitive data during runtime, even in the event of memory dumps or active attacks. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | What it is: The process of verifying the identity of a user or system.  How it’s used: Implement multi-factor authentication (MFA) using a combination of passwords, biometrics, and one-time codes. Use OAuth 2.0 for secure access tokens in APIs.  Why it applies: Ensures that only authorized users gain access to sensitive systems and data. |
| Authorization | What it is: Defining and enforcing permissions for users or systems to access specific resources.  How it’s used: Employ role-based access control (RBAC) to assign granular permissions based on roles. Ensure sensitive operations require additional approvals.  Why it applies: Reduces the risk of privilege escalation and unauthorized access. |
| Accounting | What it is: Logging and auditing all actions performed by users and systems.  How it’s used: Maintain centralized logging with tools like the ELK stack. Track actions like database modifications, file accesses, and admin commands.  Why it applies: Provides an audit trail to identify potential security incidents and support forensic analysis. |

**\***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

|  |  |  |
| --- | --- | --- |
| Standard | Principle(s) | Justification |
| STD-001-CPP | Validate Input Data | Ensures values conform to data types, reducing overflow and type mismatches. |
| STD-002-CPP | Sanitize Data Sent to Systems | Prevents invalid or malicious data from being sent to downstream processes. |
| STD-003-CPP | Practice Defense in Depth | Ensures multiple safeguards exist to handle string-related vulnerabilities. |
| STD-004-CPP | Sanitize Data Sent to Systems | Validates SQL queries to prevent injection vulnerabilities. |
| STD-005-CPP | Adhere to Least Privilege | Limits memory access to authorized components, reducing exposure to exploits. |
| STD-006-CPP | Use Effective QA Techniques | Validates assumptions during debugging, ensuring secure development practices. |
| STD-007-CPP | Keep It Simple | Simplifies exception handling, making errors predictable and less prone to misuse. |
| STD-008-CPP | Validate Input Data | Checks input correctness to reduce the likelihood of exploitation. |
| STD-009-CPP | Practice Defense in Depth | Manages resources securely to prevent leaks or misuse. |
| STD-010-CPP | Adopt a Secure Coding Standard | Ensures secure logging practices, avoiding accidental exposure of sensitive data. |

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 | 12/05/2024 | Initial Template | Brad Mills |  |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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