**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 inputs are validated for correctness, type, format, and length to prevent injection attacks and ensure data integrity. |
| 1. Heed Compiler Warnings | Pay attention to compiler warnings and address them promptly to identify potential vulnerabilities and issues during the build process. |
| 1. Architect and Design for Security Policies | Incorporate security principles in the architecture and design phase, ensuring the system adheres to established security policies. |
| 1. Keep It Simple | Design and implement systems with simplicity in mind, avoiding unnecessary complexity that can introduce vulnerabilities. |
| 1. Default Deny | Configure systems to deny access by default and explicitly grant permissions only when necessary. |
| 1. Adhere to the Principle of Least Privilege | Assign only the minimum privileges required to perform a function, reducing the potential impact of a security breach. |
| 1. Sanitize Data Sent to Other Systems | Cleanse data to remove or encode malicious input before sending it to other systems or subsystems. |
| 1. Practice Defense in Depth | Use multiple layers of security to protect assets, ensuring that a single failure does not lead to system compromise. |
| 1. Use Effective Quality Assurance Techniques | Employ rigorous testing methods to identify vulnerabilities and ensure code reliability before deployment. |
| 1. Adopt a Secure Coding Standard | Follow established secure coding guidelines to minimize vulnerabilities and improve maintainability. |

### 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** | **Use Precise Data Types** |
| --- | --- | --- |
| **Data Type** | STD-001-CPP | Using precise data types prevents unexpected behavior caused by type mismatches or overflows. |

| **Noncompliant Code** |
| --- |
| The int type might not handle large numbers, leading to undefined behavior. |
| int result = 100000 \* 100000; // May overflow |

| **Compliant Code** |
| --- |
| Using long long ensures sufficient space for the calculation. |
| long long result = 100000LL \* 100000LL; // Prevents overflow |

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

| **Principles(s):**  Principle 1: Validate Input Data - Using precise data types is part of validating the correctness and type of the data, thus preventing overflows and other issues.    Principle 4: Keep it Simple - By using the correct data type the program will be simpler to understand and avoid vulnerabilities. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| SonarQube | 9.9 | "cwe\_682" | Detects implicit or explicit conversions between different numeric types that can lead to data loss, incorrect calculations, or unexpected behavior due to size mismatches or overflow conditions. |
| SonarLint | 8.8.0.11278 | "s2147" | Identifies implicit casts that might result in a loss of precision or overflow. |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Validate Input Ranges** |
| --- | --- | --- |
| **Data Value** | STD-002-CPP | Validating data ensures only acceptable values are processed, reducing risk of vulnerabilities. |

| **Noncompliant Code** |
| --- |
| No validation for userInput may lead to out-of-bounds access. |
| int index = userInput; // No range check  array[index] = value; |

| **Compliant Code** |
| --- |
| Adding a range check prevents out-of-bounds access. |
| if (userInput >= 0 && userInput < arraySize) {  array[userInput] = value;  } |

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

| **Principles(s):**  Principle 1: Validate Input Data - Validating input is essential to prevent out-of-bounds errors or injection attacks.    Principle 5: Default Deny - By default input should be denied unless it meets the correct criteria. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| SonarQube | 9.9 | cwe\_125 | Detects out-of-bounds access that may result in program crashes or other undefined behaviors due to unchecked index values. |
| SonarLint | 8.8.0.11278 | s4664 | Highlights array accesses where the index might exceed the array’s bounds. |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Use Safe String Operations** |
| --- | --- | --- |
| **String Correctness** | STD-003-CPP | Avoids buffer overflows by limiting string operations to safe functions. |

| **Noncompliant Code** |
| --- |
| Direct use of strcpy may overflow the buffer. |
| char buffer[10];  strcpy(buffer, userInput); // No bounds checking |

| **Compliant Code** |
| --- |
| Using strncpy with size constraints prevents overflows. |
| char buffer[10];  strncpy(buffer, userInput, sizeof(buffer) - 1);  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):**  Principle 1: Validate Input Data - String operations need input validation to avoid buffer overflows.    Principle 7: Sanitize Data Sent to Other Systems - String data being sent should be sanitized. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| SonarQube | 9.9 | cwe\_120 | Detects use of unsafe string copy operations without size limits |
| SonarLint | 8.8.0.11278 | s2130 | Identifies use of functions susceptible to buffer overflow attacks. |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Use Parameterized Queries** |
| --- | --- | --- |
| **SQL Injection** | STD-004-CPP | Protects databases by ensuring user inputs are not directly included in SQL commands. |

| **Noncompliant Code** |
| --- |
| Directly appending user input makes the query susceptible to injection. |
| string query = "SELECT \* FROM users WHERE id = " + userInput;  executeQuery(query); |

| **Compliant Code** |
| --- |
| Parameterized queries sanitize inputs to prevent injection. |
| string query = "SELECT \* FROM users WHERE id = ?";  preparedStatement.setInt(1, userInput);  preparedStatement.execute(); |

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

| **Principles(s):**  Principle 1: Validate Input Data - Validating inputs prevent malicious injection attacks.    Principle 7: Sanitize Data Sent to Other Systems - SQL databases are external systems where data should be sanitized before it is transferred. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| SonarQube | 9.9 | cwe\_89 | Detects situations where input is directly used in a SQL query string without sanitization or parameterization, making SQL Injection likely |
| SonarLint | 8.8.0.11278 | s3649 | Identifies dynamically constructed SQL queries that could be vulnerable to injection |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Use Smart Pointers** |
| --- | --- | --- |
| **Memory Protection** | STD-005-CPP | Ensures proper memory management, avoiding leaks and double-frees. |

| **Noncompliant Code** |
| --- |
| Manual memory management can lead to double-frees. |
| int\* ptr = new int(5);  delete ptr;  delete ptr; // Double delete |

| **Compliant Code** |
| --- |
| Smart pointers automatically manage memory lifecycle. |
| std::unique\_ptr<int> ptr = std::make\_unique<int>(5); |

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

| **Principles(s):**  Principle 4: Keep It Simple - Smart pointers simplify memory management reducing the chance of error.    Principle 9: Use Effective Quality Assurance Techniques - Using smart pointers improves code quality by preventing errors. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| SonarQube | 9.9 | cwe\_401 | Detects memory leaks and improper management by ensuring dynamic memory allocations are freed. |
| SonarLint | 8.8.0.11278 | s2321 | Suggests using RAII (Resource Acquisition Is Initialization) principles to avoid memory leaks. |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Validate Assumptions with Assertions** |
| --- | --- | --- |
| **Assertions** | STD-006-CPP | Assertions verify assumptions during development, identifying unexpected conditions early. |

| **Noncompliant Code** |
| --- |
| Skipping validation can propagate invalid states. |
| int result = calculate();  // No assertion to validate result |

| **Compliant Code** |
| --- |
| Adding assertions ensures validity during development. |
| int result = calculate();  assert(result >= 0); |

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

| **Principles(s):**  Principle 2: Heed Compiler Warnings - Assertions function to discover problems early and give warnings in development.    Principle 9: Use Effective Quality Assurance Techniques - Assertions are part of a thorough quality assurance process. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Medium | Low | Medium | 1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| SonarQube | 9.9 | cwe\_617 | Detects reachable assertions that may result in a denial of service and identifies where asserts might be missing in a system. |
| SonarLint | 8.8.0.11278 | s3946 | Identifies instances of assertions used as a means of application control. |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Handle Exceptions Properly** |
| --- | --- | --- |
| **Exceptions** | STD-007-CPP | Proper exception handling prevents crashes and improves program reliability. |

| **Noncompliant Code** |
| --- |
| Catching all exceptions without handling can mask errors. |
| try {  riskyOperation();  } catch (...) {  // No error-specific handling  } |

| **Compliant Code** |
| --- |
| Handling specific exceptions ensures better diagnostics. |
| try {  riskyOperation();  } catch (const SpecificException& e) {  logError(e.what());  } |

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

| **Principles(s):**  Principle 2: Heed Compiler Warnings - Exception handling prevents unexpected behavior.    Principle 8: Practice Defense in Depth - Exception handling is one component of a defensive strategy to prevent compromise. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| SonarQube | 9.9 | cwe\_703 | Detects unhandled exceptions which may result in a crash or inconsistent state. |
| SonarLint | 8.8.0.11278 | s2472 | Suggests adding code to either log or handle an exception instead of silently catching and ignoring. |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Validate File Operations** |
| --- | --- | --- |
| **File Input/Output Safety** | STD-008-CPP | Ensures that file input/output (I/O) operations are performed securely, avoiding leaks, overwrites, or unauthorized access. |

| **Noncompliant Code** |
| --- |
| Writing to a file without validating permissions may expose sensitive data. |
| std::ofstream file("data.txt");  file << "Sensitive data"; // No validation of file permissions |

| **Compliant Code** |
| --- |
| Validating file permissions and ensuring secure modes prevent unauthorized access. |
| std::ofstream file;  file.open("data.txt", std::ios::out | std::ios::trunc);  if (file.is\_open() && checkFilePermissions("data.txt")) {  file << "Sensitive data";  file.close();  } |

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

| **Principles(s):**  Principle 3: Architect and Design for Security Policies - File permissions must be accounted for in design.  Principle 6: Adhere to the Principle of Least Privilege |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| SonarQube | 9.9 | cwe\_22 | Detects use of absolute paths in file I/O operations that may result in unauthorized access or modification of files. |
| SonarLint | 8.8.0.11278 | s6557 | Identifies places where file operations may occur prior to permission verification. |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Use Thread-Safe Operations** |
| --- | --- | --- |
| **Thread Safety** | STD-009-CPP | Prevents race conditions and ensures consistency when multiple threads access shared resources. |

| **Noncompliant Code** |
| --- |
| Accessing shared variables without synchronization can cause race conditions. |
| int counter = 0;  void increment() {  counter++; // Unsafe in a multithreaded context  } |

| **Compliant Code** |
| --- |
| Using a mutex ensures safe access to shared resources in a multithreaded environment. |
| std::mutex counterMutex;  int counter = 0;  void increment() {  std::lock\_guard<std::mutex> lock(counterMutex);  counter++;  } |

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

| **Principles(s):**  Principle 4: Keep It Simple - By using thread-safe constructs, complex code is simplified.  Principle 8: Practice Defense in Depth - Thread safety adds another layer of protection to sensitive operations. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| SonarQube | 9.9 | cwe\_366 | Detects the usage of non-thread safe methods or constructs in a concurrent environment. |
| SonarLint | 8.8.0.11278 | s3071 | Suggests using correct synchronization primitives and checking them. |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Use Secure Cryptographic Libraries** |
| --- | --- | --- |
| **Cryptographic Best Practices** | STD-010-CPP | Ensures sensitive data is securely encrypted and avoids vulnerabilities from using outdated or insecure algorithms. |

| **Noncompliant Code** |
| --- |
| Custom encryption algorithms are often insecure and vulnerable to attacks. |
| std::string encrypt(const std::string& input) {  // Custom "encryption" algorithm  return input + "123";  } |

| **Compliant Code** |
| --- |
| Using established libraries like OpenSSL ensures data encryption adheres to modern security standards. |
| #include <openssl/evp.h>    std::string encrypt(const std::string& input) {  // Use OpenSSL to encrypt data securely  // (Implementation details go here based on OpenSSL standards)  } |

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

| **Principles(s):**  Principle 4: Keep It Simple - By using tried and tested cryptographic libraries, complex code is simplified.    Principle 8: Practice Defense in Depth - Cryptography is a key component in layered security. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| SonarQube | 9.9 | cwe\_327 | Detects usage of insecure cryptographic algorithms, missing IVs, incorrect key lengths or usage of deprecated algorithms. |
| SonarLint | 8.8.0.11278 | s5547 | Suggests avoiding any usage of hard coded values in any part of cryptographic process |
| [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 for enforcement and compliance with this security policy will leverage SonarQube as the primary code analysis platform. SonarQube will be integrated into our pipeline to ensure that code is automatically scanned for vulnerabilities during the build process. In the design and build process SonarLint will notify developers about potential issues directly within their IDEs during development, providing instant feedback. Developers can then address these issues before committing their code. SonarQube provides rule customization, code quality metrics, and long-term tracking of vulnerabilities and technical debt. This ensures a consistent, automated approach to security across our codebase. Reports generated by both tools will be used to create a central hub where reports and security issues can be monitored and reviewed. This will allow for quicker identification of issues that were not caught earlier in the pipeline.

### 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 | High | Medium | High | 4 |
| STD-003-CPP | High | High | Medium | High | 4 |
| STD-004-CPP | High | Medium | Medium | High | 5 |
| STD-005-CPP | Medium | Medium | Low | Medium | 3 |
| STD-006-CPP | Low | Medium | Low | Medium | 1 |
| STD-007-CPP | Medium | Medium | Low | Medium | 2 |
| STD-008-CPP | High | Medium | Medium | High | 4 |
| STD-009-CPP | High | Medium | High | High | 4 |
| STD-010-CPP | High | High | Medium | High | 5 |
| [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 | Data is encrypted on disk using AES-256, with key management systems. This protects against physical data theft. Policy applies to databases, file systems, and backups. |
| Encryption in flight | Data is encrypted using TLS 1.3 when transmitted over networks. This ensures data integrity and confidentiality. Policy applies to all external APIs and internal application communications. |
| Encryption in use | Sensitive data in active memory will be encrypted using tokenization and Homomorphic encryption techniques. These protect against memory dump attacks. Policy applies to highly sensitive data like user credentials. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | We use multi-factor authentication for all access points. We require passwords to be at least 12 characters with a mix of characters. Passwords must be changed every 90 days. We will review any access request monthly and adjust the user levels as needed. |
| Authorization | Access is determined by least privilege and role-based access control (RBAC). We will maintain a permission audit and adjust permissions as users move into different positions and have their roles change. |
| Accounting | All access requests, changes to the database, additions of new users, level of access, and files accessed by users will be recorded in logs and monitored for anomalies. All logs will be retained for a period of 12 months and be used to discover any unauthorized access. |

**\***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 | 11/27/2024 | MileStone | James Hagan | James Hagan |
| 1.2 | 12/15/2024 | Finale | James Hagan | James Hagan |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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